iVIEW: infra-estrutura de visualização de informação evolucionária na web

Adriane Kaori Oshiro

Orientadora: *Profa. Dra. Maria da Graça Campos Pimentel*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação.

“VERSÃO REVISADA APÓS A DEFESA”

Data da Defesa: 19/03/2003

Visto do Orientador:

USP – São Carlos
Março/2003

Aos meus pais.

Agradecimentos

A *Deus*, por estar presente em todos os momentos.

Aos meus pais *João* e *Kazuko* pelo apoio incondicional e pela força que sempre representaram.

Aos meus familiares pelo incentivo e pelo apoio dado de todas as formas possíveis.

À minha professora e orientadora *Maria da Graça C. Pimentel* pelos ensinamentos, pelo apoio e incentivo, pela confiança depositada e, acima de tudo, pela amizade.

Aos docentes do ICMC pelo incentivo e pelos conhecimentos transmitidos.

Aos colegas do Internímia *Alessandra, Andréa, Cláudia, Juninho, Renato* e todos os demais pelo apoio oferecido durante a realização deste trabalho e pela amizade.

Aos demais colegas, especialmente os velhos *companheiros de guerra* da graduação, pela amizade.

Aos funcionários da sessão acadêmica e dos laboratórios de computação pelo auxílio e pelos serviços prestados.

Ao CNPq pelo apoio financeiro.

E a todos que me incentivaram, de uma maneira ou de outra, meus sinceros agradecimentos.

Índice

Índice de Figuras	iii
Resumo	v
Abstract	vi
Capítulo 1 - Introdução	1
1.1 CONTEXTO	1
1.2 MOTIVAÇÃO	2
1.3 OBJETIVOS	4
1.4 RESULTADOS	4
1.5 ORGANIZAÇÃO DA DISSERTAÇÃO	5
Capítulo 2 - Visualização de Informação	6
2.1 CONSIDERAÇÕES INICIAIS	6
2.2 VISUALIZAÇÃO E A TAXONOMIA DE SHNEIDERMAN	6
2.3 VISUALIZAÇÃO DE DADOS NÃO TEMPORAIS	7
2.3.1 Dados Unidimensionais	7
2.3.2 Dados Bidimensionais	8
2.3.3 Dados Tridimensionais	9
2.3.4 Dados Multidimensionais	9
2.3.5 Dados Hierárquicos	11
2.3.6 Dados com Estrutura de Rede	13
2.4 VISUALIZAÇÃO DE DADOS TEMPORAIS	13
2.4.1 Perspective Wall	13
2.4.2 LifeLines	14
2.4.3 LifeStreams	15
2.4.4 MMVIS	15
2.4.5 FilmFinder	16
2.4.6 Time Tube	17
2.5 CONSIDERAÇÕES FINAIS	18
Capítulo 3 - Visualização de Informação em Ambientes de Captura e Acesso	19
3.1 CONSIDERAÇÕES INICIAIS	19
3.2 ECLASS	19
3.3 TEAMSPACE	22
3.4 THE CONFERENCE ASSISTANT	23
3.5 DUMMBO	26
3.6 NOTEPALS	28

3.7 CONSIDERAÇÕES FINAIS	29
Capítulo 4 – Padrões XML	30
4.1 CONSIDERAÇÕES INICIAIS	30
4.2 XML	31
4.3 XML SCHEMA	32
4.4 SVG	34
4.5 XSLT	36
4.6 PROCESSADORES XSLT	38
4.6 CONSIDERAÇÕES FINAIS	39
Capítulo 5 – Infra-estrutura iVIEW	41
5.1 CONSIDERAÇÕES INICIAIS	41
5.2 VISÃO GERAL DA INFRA-ESTRUTURA IVIEW	42
5.3 ELEMENTOS E ESTRUTURAS DE VISUALIZAÇÃO DA IVIEW	44
5.3.1 Estrutura Básica de Visualização	44
5.3.2 Grupos de Elementos de Visualização	45
5.3.3 Filtros	47
5.4 ETAPAS PARA GERAÇÃO DA REPRESENTAÇÃO GRÁFICA DE INFORMAÇÕES EM SVG	48
5.4.1 Etapa 1: Geração do Documento de Transformação	48
5.4.2 Etapa 2: Geração da Representação SVG	55
5.4.3 Etapa 3: Geração do Documento com Funções de Interação	58
5.5 CONSIDERAÇÕES FINAIS	61
Capítulo 6 – Exemplo de Utilização da iVIEW no Ambiente iClass	64
6.1 CONSIDERAÇÕES INICIAIS	64
6.2 REPRESENTAÇÃO SVG DE INFORMAÇÕES EXTRAÍDAS DO AMBIENTE ICLASS	64
6.3 CONSIDERAÇÕES FINAIS	70
Capítulo 7 – Conclusões	71
Referências Bibliográficas	73
Apêndice A	77
Apêndice B	82
Apêndice C	88

Índice de Figuras

Figura 1: Uma espiral permite representar o crescimento contínuo e concêntrico da informação evolucionária [Pimentel et al., 2000b].	4
Figura 2: Interface do WebBook [Card et. al, 1996].	7
Figura 3: Código de Software exibido através do SeeSoft [Eick et al., 1992].	8
Figura 4: Interface do Information Mural [Jerding & Stasko, 1995].	8
Figura 5: A interface do Visible Human Explorer [Shneiderman et al., 1996a].	9
Figura 6: Tela do Inxight Eureka.	10
Figura 7: Tela do Cone Tree [Carriere & Kazman, 1995].	11
Figura 8: Visualização de dados de alunos através do Treemap – o tamanho dos retângulos varia de acordo com a média final do aluno. As cores indicam o conceito obtido (A, B ou C).	12
Figura 9: Exemplo de Perspective Wall [Mackinlay et al., 1991].	14
Figura 10: Exemplo de um relatório médico apresentado através do LifeLines [Shneiderman et al., 1996b].	14
Figura 11: LifeStreams dispõe documentos pessoais em ordem cronológica [Freeman & Gelernter, 1996].	15
Figura 12: Exemplo de ambiente MMVIS [Hibino & Rundensteiner, 1996].	16
Figura 13: Exemplo de FilmFinder [Shneiderman & Ahlberg, 1994].	16
Figura 14: Exemplo de Time Tube [Chi et al., 1998].	17
Figura 15: Interface Web de uma das aulas capturadas no ambiente eClass.	21
Figura 16: Tela do MeetingViewer. O MeetingViewer pode ser utilizado para visualizar uma única reunião ou um conjunto de reuniões.	23
Figura 17: À esquerda, a parte frontal do DUMMBO, sem botões, câmeras, ou telas de computador. À direita, a parte de trás do DUMMBO, onde ficam “escondidos” os equipamentos eletrônicos, como computadores e microfones.	26
Figura 18: Interface de acesso à informação capturada do DUMMBO.	27
Figura 19: Tela do Note Browser.	28
Figura 20: Árvore SVG implementada por Southard ¹³ . Ao posicionar o mouse sobre uma folha, o nome do atributo correspondente e seu conteúdo são mostrados.	34
Figura 21: Winter Tree associada a um site de autoria colaborativa.	35
Figura 22: Exemplo de um <i>piechart</i> gerado no AgileBlox da Elansoft.	36
Figura 23: Com o Saxon, utilizando-se documentos XML e XSLT (pode-se também combinar vários documentos XSLT) como entrada, é possível gerar outros tipos de documentos como HTML, SVG, JavaScript, ou ainda outros documentos XML e XSLT.	39
Figura 24: Informações exibidas através de uma representação SVG utilizando <i>plug-in</i> para browser	39

Figura 25: Etapas para obtenção de uma visualização de informações em SVG. Os Documentos 1 e 2 são criados pelo desenvolvedor. O restante dos documentos são providos pela infra-estrutura iVIEW (3, 4 e 7) ou gerados automaticamente (5, 6 e 8). As linhas duplas que unem o Documento 2 ao Documento 3 significa que o Documento 2 deve obedecer as especificações contidas no Documento 3.	43
Figura 26: Exemplo de configuração da estrutura básica de visualização do iVIEW – disposição vertical das áreas principais.	45
Figura 27: Exemplo de configuração da estrutura básica de visualização do iVIEW – disposição horizontal das áreas principais.	45
Figura 28: Grupos de elementos XML distintos.	46
Figura 29: Filtro de seleção de elementos. a) Todos os alunos de todos os cursos são exibidos. b) Apenas os alunos de “Computação” e “Física” são exibidos.	48
Figura 30: Apresentação de elementos correspondentes a referências bibliográficas, conforme definidos na instância XML que especifica o formato da visualização final.	51
Figura 31: Formulário para geração de uma instância XML para especificação de layout, elementos e filtros (Documento 2 da Figura 25).	54
Figura 32: Representação SVG de informações sobre aulas capturadas.	58
Figura 33: As informações sobre um determinado elemento são representadas por atributos especificados pelo desenvolvedor.	59
Figura 34: Exemplo de relacionamento entre elementos.	60
Figura 35: Exemplos de filtros. a) Dois filtros atuando sobre um único grupo de elementos. b) Um único filtro atuando sobre dois grupos de elementos.	61
Figura 36: Representação SVG associada ao documento JavaScript contendo funções que provêem interação com os elementos apresentados.	61
Figura 37: Instância do documento XML de sessões capturadas (session.xml).	65
Figura 38: Representação SVG de sessões capturadas no iClass, do 2º. semestre de 2002 ao 2º. semestre de 2003	68
Figura 39: Representação SVG de sessões capturadas após a interação de um usuário com funções contidas em um documento JavaScript.	69

Resumo

O crescimento acelerado da Internet como meio de se obter informações dentro dos mais diversos domínios de aplicações torna necessária a incorporação de técnicas visuais que auxiliem o usuário a interagir com esse vasto universo de informações de forma eficaz e intuitiva. A **Visualização de Informação** é uma área de pesquisa que estuda o uso de representações visuais e interativas de informações com o objetivo de diminuir a sobrecarga cognitiva dos usuários ao interpretá-las.

Os objetivos deste trabalho são investigar e implementar mecanismos que permitem ao usuário utilizar a Web para visualizar **informações evolucionárias** – informações que crescem ao longo do tempo e se relacionam. Diversas técnicas de visualização existentes foram pesquisadas. Foi investigada a utilização de padrões XML (Extensible Markup Language) para geração de representações gráficas através de documentos SVG (Scalable Vector Graphics). Como resultado, foi definida a infra-estrutura iVIEW – infra-estrutura de **Visualização de Informação Evolucionária na Web** – que provê um mecanismo para visualização de informação evolucionária através do processamento de documentos estruturados.

Abstract

The recent growth of the Internet as a way to get information in the context of several application domains has demanded the incorporation of visual techniques that aid users in the task of interacting with this vast universe of information in an efficient and intuitive manner. **Information Visualization** is a research area that investigates the use of visual and interactive representations of information with the aim of reducing users' cognitive overhead as they analyse information.

The main goals of this work are the investigation and implementation of mechanisms that allow users to use the Web to visualize **evolutionary information** – information that grows along time and is related to other information. Several visualization techniques have been investigated. The use of XML (Extensible Markup Language) standards to generate graphic representations through SVG (Scalable Vector Graphics) documents was also investigated. As a result, we have defined the iVIEW – infrastructure for the visualization of evolutionary information in the Web, which provides a mechanism for evolutionary information visualization through structured documents processing.

Capítulo 1 - Introdução

1.1 CONTEXTO

Nos últimos anos, o aumento da velocidade de processamento e da resolução das telas de computadores tem desempenhado um papel fundamental para o aprimoramento de interfaces gráficas para a interação do usuário com a informação apresentada. O subsequente crescimento da Internet e a grande quantidade e variedade de informações disponíveis, foram outros fatores que tornaram necessária a incorporação de técnicas visuais que auxiliem o usuário a interagir com esse vasto universo de informações de forma eficaz e intuitiva.

Entretanto, representar a informação não é uma tarefa simples, pois a quantidade de informações disponíveis na *Web* é relativamente grande e evolui com frequência e rapidez. Conseqüentemente, a tarefa de localizar ou interpretar as informações que procura pode tornar-se lenta e desgastante para o usuário. Além disso, grande parte das informações é de natureza abstrata, ou seja, corresponde a dados que só existem na forma de textos, números, gráficos e tabelas, pois não são objetos físicos como, por exemplo, peças que compõem um automóvel [Card et al., 1999]. Dados estatísticos, conteúdo de disciplinas, relatórios administrativos, entre outros, são exemplos de dados abstratos.

Torna-se importante a existência de mecanismos com os quais o usuário possa:

- ❑ Localizar dados de forma rápida e precisa;
- ❑ Interpretar os dados de maneira intuitiva;
- ❑ Visualizar os diversos atributos de um determinado item de informação e as relações que existem entre eles;
- ❑ Tomar decisões a partir dos dados apresentados, uma vez que se todo um conjunto de informações puder ser visualizado numa única tela, por exemplo, haverá maior facilidade em identificar o relacionamento entre eles.

Para que se pudesse aplicar técnicas de visualização em diversos tipos de aplicações, a **Visualização de Informação** surgiu como uma área de pesquisa que estuda o uso de representações visuais e interativas de dados abstratos suportados por computador diminuindo, dessa forma, a sobrecarga cognitiva do usuário ao interpretar esses dados [Card et al., 1999].

1.2 MOTIVAÇÃO

Um grupo de pesquisadores do Georgia Institute of Technology (GATECH) tem desenvolvido, desde 1995, o projeto do ambiente *eClass*¹, no qual é investigada a utilização de computação ubíqua para a captura de informações em ambientes de sala de aula. Um dos objetivos do projeto é permitir a produção automática de documentos hipermídia para a Web que rellitam o conteúdo capturado [Abowd, 1999] [Abowd et al., 1998a, b] [Brotherton & Abowd, 1998] [Brotherton et al., 1998]. [Pimentel & Abowd 1999] propõem a modelagem dos processos de armazenamento, recuperação, extensão e **visualização da informação** obtida a partir da captura automática de atividades interativas em ambientes instrumentados com tecnologias computacionais.

O projeto InCA-SERVE, do Programa de Cooperação Internacional entre ProTeM-CC/CNPq/Brasil e NSF/EUA e também apoiado pela FAPESP, pretende desenvolver infra-estruturas de captura, acesso, armazenamento, extensão, recuperação e **visualização** de informação evolucionária capturada [Pimentel & Abowd, 1999] [Pimentel, 2000]. Essas infra-estruturas são denominadas InCA e SERVE.

Apesar de os grupos de pesquisa do GATECH e do ICMC-USP estarem tratando dos problemas relacionados à captura, armazenamento e recuperação das informações, existe ainda uma lacuna no que se refere à **visualização** dessas informações, pois as técnicas de visualização que existem atualmente não consideram o crescimento evolucionário das informações.

¹ Anteriormente denominado Classroom 2000 (<http://www.cc.gatech.edu/lce/eclass/index.html>).

É importante notar que as informações capturadas, no caso de uma experiência em sala de aula, fazem parte de um contexto que abrange informações produzidas antes e depois da aula em si:

- **Antes da aula**, o professor prepara material a ser apresentado e/ou discutido;
- **Durante a aula**, professor e alunos podem fazer anotações em material produzido anteriormente ou produzir novo material;
- **Após a aula**, professor e alunos podem visitar o conteúdo discutido durante a aula.

Deve-se considerar ainda que cada nova aula está intrinsecamente relacionada com outras que já ocorreram, e o material capturado, como um todo, reflete uma constante evolução. Nesse contexto, é necessário que a visualização das informações geradas a partir da captura de diversas aulas seja capaz de expressar o relacionamento intrínseco existente entre elas e o seu crescimento evolucionário.

Em geral, identifica-se duas características importantes associadas a um conjunto de sessões capturadas [Pimentel et al. 2000b]:

- **Expansão progressiva**: o repositório de informações se expande continuamente durante o tempo da aplicação, como resultado da contribuição de seus participantes. Essas contribuições ocorrem antes, durante e depois de cada sessão capturada.
- **Referência intrínseca**: o material gerado em qualquer uma das três fases (antes, durante e depois da captura) pode referenciar o material gerado anteriormente no contexto da aplicação que utiliza o ambiente de captura.

Essas características são responsáveis pela complexidade das tarefas associadas à manipulação desse tipo de informação, entre elas o armazenamento, a recuperação e a **visualização** dos dados capturados. Esses problemas podem ser investigados por sistemas de captura com suporte automático à autoria de hiperdocumentos associados. No entanto, mecanismos de visualização devem considerar que o repositório de informação capturada pode ser associado a uma espiral, como ilustrado na **Figura 1**: partindo da origem a informação cresce continuamente e pode ser relacionada com a informação gerada anteriormente. Assim, referimo-nos à informação resultante de **informação evolucionária**.

Figura 1: Uma espiral permite representar o crescimento contínuo e concêntrico da informação evolucionária [Pimentel et al., 2000b].

1.3 OBJETIVOS

Os objetivos deste trabalho são investigar e implementar mecanismos que permitem a visualização da informação capturada à medida que ela evolui e sofre alterações ao longo do tempo. Nesse contexto, um mecanismo de visualização adequado seria aquele que possibilitasse ao usuário identificar como as informações crescem e como elas se relacionam. Dada a crescente importância da Web, é interessante que a mesma possa ser utilizada como plataforma de visualização dessas informações.

Para tanto, foram investigadas as diversas técnicas de visualização existentes, a fim de possibilitar o desenvolvimento de um mecanismo para visualização de **informação evolucionária**.

1.4 RESULTADOS

Como resultado, foi definida a infra-estrutura iVIEW – infra-estrutura de Visualização de Informação Evolucionária na Web – que provê um mecanismo para visualização de informação evolucionária através do processamento de documentos estruturados. Foram

utilizados documentos XML² – Extensible Markup Language – para geração de representações gráficas através de documentos SVG³ – Scalable Vector Graphics – ambos recomendações do World Wide Web Consortium.

Apesar de encontrarmos informações evolucionárias em diversos ambientes, foi utilizado um ambiente educacional, que no caso é o ambiente iClass, uma nova versão do ambiente eClass desenvolvido no contexto do Projeto InCA-SERVE como primeiro estudo de caso para a infra-estrutura de visualização implementada neste trabalho.

1.5 ORGANIZAÇÃO DA DISSERTAÇÃO

O restante desta dissertação é organizado como segue: no Capítulo 2 são apresentados conceitos sobre Visualização de Informação e descreve as características de diversos mecanismos e ferramentas propostos e utilizados; no Capítulo 3 é apresentada Visualização de Informação no contexto de ambientes de captura e acesso; no Capítulo 4 são apresentados os padrões de linguagem utilizados na implementação deste projeto; no Capítulo 5 é descrita a infra-estrutura construída para atingir os objetivos propostos; no Capítulo 6 é apresentado um exemplo de utilização da infra-estrutura implementada no ambiente iClass; por fim, no Capítulo 7 são apresentadas as conclusões a respeito dos resultados alcançados e trabalhos futuros.

² World Wide Web Consortium (<http://www.w3.org/xml>)

³ World Wide Web Consortium (<http://www.w3.org/svg>)

Capítulo 2 - Visualização de Informação

2.1 CONSIDERAÇÕES INICIAIS

Neste capítulo são apresentados conceitos sobre Visualização de Informação e alguns mecanismos e ferramentas existentes que serviram como base para o estudo de uma infraestrutura de visualização de informação evolucionária.

2.2 VISUALIZAÇÃO E A TAXONOMIA DE SHNEIDERMAN

Visualização é um processo cognitivo desempenhado pelo ser humano para formar uma imagem mental de um espaço de domínio. Na Ciência da Computação e da Informação, a visualização é, mais especificamente, a utilização de gráficos, imagens e outros recursos como animação e som para representar a estrutura e o comportamento dinâmico de um grande e complexo conjunto de informações abstratas [Card et al., 1999]. É, portanto, uma abordagem que difere da Visualização Científica, que enfoca a representação de objetos físicos do mundo real, como o corpo humano, peças mecânicas, moléculas, entre outros.

Entretanto, representar a informação não é uma tarefa trivial. Nesse contexto, dezenas de mecanismos para Visualização de Informação têm sido propostos e utilizados para que o usuário possa navegar por um vasto espaço de informações e localizar rapidamente os itens de seu interesse. A fim de compreender esse extenso e variado conjunto de mecanismos, Ben Shneiderman [Shneiderman, 1996] definiu uma taxonomia de Visualização de Informação de Interfaces de Usuário baseada na característica principal do tipo de dado a ser apresentado, agrupando esses dados em sete categorias: 1) dados temporais; 2) dados unidimensionais; 3) dados bidimensionais; 4) dados tridimensionais; 5) dados multidimensionais; 6) dados hierárquicos e 7) dados com estrutura de rede.

A seguir, são descritos os dados classificados de acordo com a taxonomia de Shneiderman, bem como aplicações e ferramentas de visualização que utilizam esses dados. Os dados não temporais e os dados temporais serão apresentados em seções separadas, devido à importância do último no contexto deste trabalho, que trata de informação evolucionária.

2.3 VISUALIZAÇÃO DE DADOS NÃO TEMPORAIS

2.3.1 Dados Unidimensionais

Dados unidimensionais são dados organizados de forma linear, ou seja, sequencialmente. Textos, códigos-fonte e listas de nomes e endereços são exemplos de dados lineares. O *WebBook* [Card et. al, 1996] é um exemplo de ferramenta que utiliza dados unidimensionais, usando a metáfora de um livro para representar páginas Web. Cada página no WebBook é uma página de um hiperdocumento na Web. Através de recursos de animação e gráficos tridimensionais, o usuário pode navegar pelo “livro” clicando nos botões localizados na sua parte inferior, ou clicando nos links que representam páginas externas ou internas contidos nos documentos, conforme pode ser observado na **Figura 2**.

Figura 2: Interface do WebBook [Card et. al, 1996].

Outro exemplo é o *SeeSoft*, uma ferramenta utilizada para visualizar informações com dados estatísticos associados. Na **Figura 3** é mostrado a visualização de um projeto de software através do SeeSoft. Cada bloco na figura representa um módulo do código-fonte do programa. Cada linha no bloco corresponde a uma linha de código daquele módulo. No exemplo da **Figura 3** as cores são utilizadas para indicar a frequência de execução de cada linha de código [Eick et al., 1992].

Figura 3: Código de Software exibido através do SeeSoft [Eick et al., 1992].

2.3.2 Dados Bidimensionais

Dados bidimensionais são dados que contêm dois atributos como, por exemplo, coordenadas horizontais e verticais de mapas geográficos. Projetos de *chip* de computador e plantas de prédios e construções são outros exemplos de aplicações que envolvem visualização bidimensional. O *Information Mural* [Jerding & Stasko, 1995] é uma ferramenta que apresenta em uma única tela grandes espaços de informação através de atributos visuais, como tonalidade e intensidade de cores e tamanho do pixel. Na **Figura 4** a densidade populacional e a localização geográfica de cidades são representadas através das técnicas do Information Mural.

Figura 4: Interface do Information Mural [Jerding & Stasko, 1995].

2.3.3 Dados Tridimensionais

Dados tridimensionais são dados que incorporam volume às suas coordenadas bidimensionais. Ferramentas para visualização de dados tridimensionais estão presentes na maioria das aplicações científicas e outras nas quais o usuário pode explorar representações de objetos reais através do computador. O *Visible Human Explorer* (VHE) [Shneiderman et al., 1996a] é uma ferramenta que pode utilizar grandes bancos de dados contendo imagens da anatomia humana. A interface do VHE provê uma miniatura do conjunto de dados disponíveis nesse banco de dados e, a partir deste conjunto, o usuário pode especificar e visualizar imagens de forma mais detalhada, como é mostrado na **Figura 5**.

Figura 5: A interface do Visible Human Explorer [Shneiderman et al., 1996a].

2.3.4 Dados Multidimensionais

Dados multidimensionais são dados que contêm mais de três atributos, os quais podem ser selecionados dinamicamente. O *Inxight Eureka* (ou *Table Lens*, na versão para Web) é uma ferramenta que possibilita a visualização de grandes quantidades de dados multidimensionais armazenados em uma tabela. Mesmo um número relativamente grande de itens (centenas ou até milhares) pode ser visualizado em uma única tela, pois os valores dos atributos de cada item são representados por barras horizontais [Rao & Card, 1994]. Os itens e seus atributos podem ser focalizados (como uma lente de aumento) sem que o usuário perca a noção da relação destes itens com os demais pertencentes à tabela. Essa

ferramenta⁴ foi experimentada utilizando um conjunto de dados contendo o nome e diversos outros atributos de uma turma de alunos de uma determinada disciplina de pós-graduação. Na **Figura 6** é mostrado como seria visualizado esse conjunto de dados. Os alunos podem ser agrupados de acordo com a área de pesquisa e o orientador, e ordenados de acordo com o atributo determinado pelo usuário (na **Figura 6** os alunos estão ordenados por ordem decrescente de média final). Para estabelecer esta ordenação, basta clicar no nome da coluna do atributo desejado. O comprimento das barras horizontais varia de acordo com o valor do atributo representado pela coluna. Quando os atributos possuem valores discretos (não numéricos, como o Conceito, no exemplo apresentado), as barras são apresentadas em cores distintas. Para visualizar o valor de um determinado atributo, basta clicar na linha (barra horizontal) que representa este atributo.

Figura 6: Tela do Inxight Eureka.

⁴ O Inxight Eureka está disponível para download em http://www.inxight.com/products_eu/eureka/index.html

2.3.5 Dados Hierárquicos

Dados hierárquicos são dados que apresentam uma estrutura hierárquica, onde cada nó (ou item) possui um nó pai, exceto o nó raiz. Uma árvore de diretórios de arquivos é um exemplo bastante comum de uma estrutura hierárquica. O *Cone Tree* [Robertson et al., 1991] é uma ferramenta que fornece ao usuário a visualização da estrutura global de um espaço de informação. Através de métodos de rotação, os nós mais distantes no Cone Tree podem ser trazidos para frente, permitindo que o usuário acesse as informações correspondentes ao nó selecionado. Na **Figura 7**, algumas ligações levam um nó a nós intermediários, enquanto outras levam um nó a nós-folhas.

Figura 7: Tela do Cone Tree [Robertson et al., 1991].

O *Treemap 3.0*⁵, desenvolvido por Ben Shneiderman durante a década de 90 na Universidade de Maryland, é uma ferramenta que possibilita a visualização de estruturas hierárquicas através de retângulos aninhados, desenhados em uma única tela, sendo que os retângulos mais internos representam os nós-folhas [Johnson & Shneiderman, 1991]. O número de nós-folhas que podem ser visualizados através do *Treemap* é de uma ordem de magnitude maior do que na visualização proporcionada por representações de árvores tradicionais, pois a área de visualização é otimizada.

Essa ferramenta foi experimentada com os mesmos dados de teste do Inxight Eureka (Seção 2.3.4). Na **Figura 8** é mostrado como as informações podem ser visualizadas

⁵ Uma versão para download pode ser obtida em <http://www.cs.umd.edu/hcil/treemap3>.

através do Treemap. Como no exemplo do Inxight Eureka, os alunos estão agrupados de acordo com a sua área de pesquisa e também de acordo com o seu orientador, formando assim uma estrutura hierárquica. A área e a cor dos retângulos que representam cada um dos alunos são determinadas de acordo com os atributos especificados pelo usuário. Por exemplo, o tamanho do retângulo pode ser determinado de acordo com a média final (poderia ser de acordo com a média de prova ou a média de trabalho) e a cor pode variar conforme com o conceito obtido pelo aluno.

Figura 8: Visualização de dados de alunos através do Treemap – o tamanho dos retângulos varia de acordo com a média final do aluno. As cores indicam o conceito obtido (A, B ou C).

O Treemap 3.0 também pode ser utilizado para visualizar estruturas de diretórios contendo milhares de arquivos numa única tela, com o tamanho dos retângulos variando de acordo com o tamanho dos arquivos em *bytes*.

2.3.6 Dados com Estrutura de Rede

Dados com estrutura de rede são dados que apresentam uma estrutura em forma de rede, ou seja, diferentemente da estrutura hierárquica, os nós possuem ligações com quaisquer outros nós. O *Butterfly Citation Browser* [Hearst et al., 1996] é uma ferramenta para a apresentação de artigos recuperados a partir de consultas feitas pelo usuário. Os artigos, suas referências e citações são visualizados utilizando a metáfora de uma borboleta. Na cabeça da borboleta estão os resumos dos artigos, na asa esquerda estão as referências e na asa direita as citações. Os itens selecionáveis e os componentes das asas da borboleta são os artigos recuperados.

2.4 VISUALIZAÇÃO DE DADOS TEMPORAIS

Mecanismos e ferramentas de visualização temporais são utilizados para visualizar dados que possuem começo e fim em relação ao tempo. Através dessas ferramentas, os usuários podem acessar as informações correspondentes a um determinado período, por exemplo. Uma das características da informação evolucionária é, além dos dados cronológicos, a constante alteração que sofre ao longo do tempo. Nesse contexto, diversos pesquisadores têm concentrado esforços para desenvolver um mecanismo que forneça a visualização desse tipo de informação, de forma a reduzir a sobrecarga cognitiva do usuário ao identificar tais alterações.

2.4.1 Perspective Wall

Perspective Wall é uma ferramenta para apresentação de dados estruturados sequencialmente, em ordem cronológica ou alfabética [Mackinlay et al., 1991]. Na **Figura 9** são mostrados dados dispostos em ordem cronológica. No centro do *Perspective Wall* os dados podem ser visualizados com mais detalhes sem que o usuário perca o contexto da informação, pois o restante dos dados é apresentado nas laterais, em menor escala.

2.4.3 LifeStreams

Através da ferramenta *LifeStreams*, documentos contendo informações sobre seus usuários são dispostos de forma linear, numa sequência cronológica [Freeman & Gelernter, 1996], como é mostrado na **Figura 11**. A ferramenta utiliza a metáfora de pilha de folhas de papel para representar informações. Cada folha de papel pode ser destacada utilizando-se o mouse.

Figura 11: LifeStreams dispõe documentos pessoais em ordem cronológica [Freeman & Gelernter, 1996].

2.4.4 MMVIS

O ambiente MMVIS (*MultiMedia Visual Information Seeking*) tem como objetivo suportar a tarefa de análise de vídeo [Hibino & Rundensteiner, 1996]. Subconjuntos especializados como filtros de consulta dinâmica temporais, espaciais e de movimento estão acoplados a visualizações dinâmicas e configuráveis para auxiliar usuários na descoberta de padrões e tendências dos dados. Os usuários podem selecionar dois subconjuntos e então visualizar os vários relacionamentos entre esses subconjuntos. Na **Figura 12** é mostrado um conjunto de relacionamentos entre os subconjuntos **A** e **B**.

Figura 12: Exemplo de ambiente MMVIS [Hibino & Rundensteiner, 1996].

2.4.5 FilmFinder

FilmFinder é uma ferramenta para visualização de dados sobre filmes [Shneiderman & Ahlberg, 1994], representados por retângulos coloridos que, quando selecionados, disponibilizam informações sobre um determinado filme, como pode ser observado na Figura 13. Ao longo do eixo “x”, os filmes estão distribuídos de acordo com o ano de produção e, ao longo do eixo “y”, conforme sua popularidade. Cores distintas são usadas para representar o gênero a que pertencem os filmes.

Figura 13: Exemplo de FilmFinder [Shneiderman & Ahlberg, 1994].

2.4.6 Time Tube

O *Time Tube* é uma ferramenta de visualização que organiza e apresenta a evolução de um Web site ao longo do tempo [Chi et al., 1998]. Vários atributos como frequência de acesso, caminhos percorridos pelo browser e alterações ocorridas nas diversas páginas são mapeados em uma *disk tree*, a qual representa a estrutura de ligações do Web site. Um conjunto de disk trees, por sua vez, forma um time tube. Na **Figura 14**, cada uma das quatro disk trees representa o *status* de um site durante as quatro semanas de um determinado mês, respectivamente.

Figura 14: Exemplo de Time Tube [Chi et al., 1998].

O status de uma disk tree pode ser observado através das seguintes características:

- As ligações do hiperdocumento são representadas por linhas (galhos da árvore).
- A frequência de acesso às páginas é medida pela espessura e pelo brilho das linhas.
- O estágio do ciclo de vida de cada página é representado por cores diferentes de linha:
 - Linha verde = página que foi mantida de uma disk tree para outra;
 - Linha amarela = página que foi removida;
 - Linha vermelha = página que foi adicionada.

2.5 CONSIDERAÇÕES FINAIS

Neste capítulo procurou-se mostrar a diversidade de ferramentas voltadas para Visualização de Informação. Muitas dessas ferramentas estão direcionadas para aplicações que utilizam conjuntos de dados específicos para situações específicas. No próximo capítulo, são apresentados ambientes de captura e acesso de informação e os recursos que esses ambientes utilizam para visualizar os dados capturados.

Capítulo 3 - Visualização de Informação em Ambientes de Captura e Acesso

3.1 CONSIDERAÇÕES INICIAIS

Tendo visto, no capítulo anterior, diversos mecanismos de visualização de informação dentro de um contexto mais geral, este capítulo aborda o que existe hoje em termos de visualização em ambientes de captura e acesso de informações. São descritos mecanismos e infra-estruturas utilizados para apresentar a informação capturada em ambientes instrumentados, como exemplos de aplicações que geram informação evolucionária. As infra-estruturas descritas são:

- *eClass*;
- *TeamSpace*;
- *Conference Assistant*;
- *DUMMBO*;
- *e NotePals*.

3.2 ECLASS

A utilização da Web como repositório de materiais didáticos é bastante freqüente nas universidades, pois facilita consideravelmente o seu acesso pelos usuários. Professores e alunos podem recuperar o conteúdo de uma determinada disciplina a partir de qualquer lugar, a qualquer momento. As informações contidas na Web podem ser enriquecidas se acrescentarmos as experiências, as dúvidas e as discussões que normalmente ocorrem durante uma aula (armazenadas na forma de áudio e vídeo, principalmente).

O eClass é um projeto desenvolvido por um grupo de pesquisadores da GATECH que faz uso da computação ubíqua para captura e acesso de experiências em salas de aula [Abowd, 1999] [Abowd et al., 1998a, b] [Brotherton & Abowd, 1998] [Brotherton et al., 1998].

Para prover a captura automática de informações multimídia geradas no dia a dia de uma universidade, são utilizados alguns equipamentos instalados nas salas de aula. Lousas

eletrônicas, computadores, câmeras e microfones são posicionados apropriadamente para que a captura ocorra de maneira fácil e confiável, atenuando significativamente o esforço necessário para a geração dessas informações e sua posterior disponibilização na Web.

Assim, como a infra-estrutura de captura de informações no ambiente eClass faz uso de diversas tecnologias para torná-la eficiente, existe também uma preocupação com a infra-estrutura de acesso e visualização dessas informações [Pimentel et al., 2000a] [Pimentel et al., 2000b]. Devemos considerar que as informações geradas durante uma aula dentro de um ambiente de computação ubíqua possuem diversas características, entre elas:

- As diversas mídias capturadas como áudio, vídeo e anotações sobre os slides apresentados devem estar relacionadas e sincronizadas entre si;
- O repositório de informações cresce continuamente, à medida que o curso evolui, como resultado da participação de todos os envolvidos (neste caso, alunos e professores);
- O conteúdo de uma determinada aula pode estar relacionado a uma outra disciplina ou ainda fazer referências a outras fontes de informação, como uma página Web, por exemplo.

Essas características compõem o que chamamos de informação evolucionária, conforme discutido no Capítulo 1. Na **Figura 15** é mostrada a interface Web de uma das aulas capturadas no ambiente eClass.

Cada aula é composta por um conjunto de slides, que são apresentados em sequência. Atualmente os slides são armazenados na forma de imagens no formato *gif* ou *jpg*. O usuário pode acessá-los através da barra de rolagem à direita, ou seguindo a linha de tempo à esquerda da tela, onde os slides são representados por links. Essa linha de tempo apresenta também eventos relevantes que possam ter ocorrido em cada slide, como referências a páginas Web. Esses eventos também aparecem na forma de links e quando clicados levam o usuário à URL correspondente.

Devemos destacar que, para que as referências a páginas Web sejam capturadas e fiquem acessíveis na linha de tempo, as páginas Web correspondentes devem ter sido visitadas durante a sessão de captura.

Figura 15: Interface Web de uma das aulas capturadas no ambiente eClass.

As anotações escritas manualmente na lousa eletrônica (*handwriting*) podem eventualmente estar contidas nos slides. Elas desempenham uma função importante, além de detalharem o conteúdo do slide. As anotações são âncoras para o áudio e/ou vídeo correspondentes. Estes podem ser acionados a partir das anotações, caso tenham sido capturados. Esse recurso permite que o usuário possa acessar diretamente a parte do áudio e vídeo de seu interesse.

Tendo em consideração as diversas características da informação gerada durante as aulas, uma nova versão do ambiente eClass é utilizada mais adiante como estudo de caso para a infra-estrutura iVIEW implementada no contexto deste trabalho.

3.3 TEAMSPACE

Durante reuniões de trabalho convencionais, o andamento do assunto discutido é apresentado aos participantes, decisões são tomadas, alternativas são avaliadas e detalhes são esclarecidos. No entanto, a maior parte da discussão gerada nessas reuniões não fica registrada formalmente em algum documento, ficando apenas na memória dos participantes. O uso da computação ubíqua para capturar informação gerada em reuniões de trabalho vem a ser um grande auxílio para preservar grandes quantidades de dados, que podem ser muito úteis posteriormente [Richter et al., 2001].

O TeamSpace é um sistema desenvolvido em conjunto pelo centro de pesquisa da IBM, pela Boeing e pela GATECH, para suporte a ambientes colaborativos de captura de reuniões. Este sistema visa dar suporte tanto a atividades síncronas quanto assíncronas, e a sua aplicação é voltada essencialmente à Web. O sistema TeamSpace é composto por um servidor que pode ser acessado por um Web browser padrão, um grupo de clientes que estarão conduzindo e capturando as reuniões virtuais e um componente de acesso ao material capturado chamado *MeetingViewer*. Por se tratar de um mecanismo utilizado para visualizar a informação dentro do ambiente suportado pelo TeamSpace, o MeetingViewer é descrito com mais detalhes a seguir.

Após o encerramento de uma reunião, os dados capturados ficam automaticamente disponíveis para posterior consulta. Os usuários podem selecionar uma reunião no TeamSpace e visualizar o seu conteúdo através da *applet* do MeetingViewer, que é ilustrado na **Figura 16**. O MeetingViewer integra todas as informações a respeito da reunião baseadas em tempo. O usuário pode utilizar uma linha de tempo em duas escalas para navegar por um conjunto de reuniões selecionadas, podendo fazer um *playback* de qualquer uma dessas reuniões. Os eventos significativos são marcados com cores diferentes na linha de tempo, a fim de prover um sumário visual da reunião, e também para auxiliar a navegação. Por eventos significativos podemos entender como sendo a entrada e a saída de pessoas, itens da agenda sendo discutidos, itens de ação visitados ou criados e slides visitados. Os usuários podem controlar quais desses eventos eles querem ver e podem utilizá-los para encontrar porções relevantes dentro de uma reunião. A interface de

visualização não envolve exibir apenas áudio e vídeo, mas também todos os demais eventos registrados na reunião, como visitas aos slides ou discussão de itens da agenda.

Figura 16: Tela do MeetingViewer. O MeetingViewer pode ser utilizado para visualizar uma única reunião ou um conjunto de reuniões.

O restante das informações é mostrado em um conjunto de quadros⁶ sobrepostos para cada objeto relacionado à reunião, incluindo descrições e resumos da reunião, agenda, apresentações, itens de ação e imagens de vídeo. Estes quadros são uma abordagem genérica para exibição de uma grande quantidade de informações relacionadas entre si. Entretanto, para simplificar a visualização, cada quadro pode ser aberto em uma janela separada, movido e redimensionado. Desta forma, os usuários podem visualizar qualquer subconjunto da informação, a curto prazo, que desejam acessar.

3.4 THE CONFERENCE ASSISTANT

O *Conference Assistant* é uma aplicação para captura e acesso de informação gerada em apresentações distribuídas, que foi projetado para auxiliar as pessoas que assistem a sessões de conferências. Existe uma grande variedade de atividades em conferências, como

⁶ Na Figura 16 o quadro exibido é o de apresentações (Presentations).

apresentação de artigos, demonstrações, reuniões especiais de grupos com interesses comuns, entre outros. O Conference Assistant também visa auxiliar os usuários a decidir de quais atividades participar, a estar ciente das atividades realizadas por outros usuários, a tomar nota das apresentações e a recuperar informação após o encerramento da conferência [Dey et al., 1999].

Os diversos recursos providos pelo Conference Assistant podem ser melhor ilustrados através do exemplo a seguir, dentro do contexto de uma conferência:

- 1) Ao chegar à conferência, o usuário se registra na máquina onde está o servidor central, fornecendo informações para contato, como endereço, telefone e e-mail, uma lista de pesquisas de seu interesse e uma lista de colegas que também estão participando da conferência.
- 2) A partir de então o usuário passa a ter acesso ao Conference Assistant em seu computador pessoal (em geral um laptop). Ao iniciar a aplicação, esta exibe automaticamente uma cópia da programação da conferência, mostrando tanto as sessões de apresentação de artigos quanto as de demonstrações.
- 3) Na tela da programação da conferência que é exibida ao usuário, algumas palestras e demonstrações aparecem em destaque para indicar que elas podem ser de interesse particular ao usuário.
- 4) Escolhida a sala onde se realizará a apresentação de seu interesse, o Conference Assistant exibe o nome do palestrante e o título da apresentação, informando também se áudio e/ou vídeo estão sendo capturados.
- 5) Uma miniatura do slide ou página Web que está sendo apresentado pelo palestrante é exibida na tela do usuário. Nessa tela o usuário pode escrever suas observações e anexá-las aos slides ou páginas Web correspondentes.
- 6) Após a apresentação, o usuário verifica novamente a programação da conferência e percebe que o Conference Assistant sugeriu uma sessão de demonstração baseado em seus interesses. O usuário dirige-se então para o local onde se realizará a demonstração.
- 7) À medida que o usuário caminha pelas sessões de demonstração à procura daquela de seu interesse, a aplicação exibe na tela do usuário o nome de cada apresentador e sua respectiva demonstração.

- 8) Escolhida a demonstração, o usuário pode informar o nível de seu interesse para a aplicação. Através dela o usuário pode localizar seus colegas em outras apresentações e observar se algum deles apresenta alto nível de interesse em alguma apresentação em particular.
- 9) Dias após a conferência, o usuário pode recuperar informações a respeito de alguma apresentação em particular. Para isso a aplicação exibe uma linha de tempo da programação da conferência com todas as sessões de palestras e demonstrações.
- 10) Através de uma interface de busca, o Conference Assistant permite que o usuário visualize a linha de tempo contendo vários eventos: sua chegada e saída das diversas salas de conferência; as questões que ele levantou durante as apresentações de que participou; as questões levantadas por outros usuários; as palavras-chaves utilizadas, entre outros. Selecionando um desses eventos o usuário pode acessar o slide ou página Web que estava sendo apresentado no momento em que ocorreu o evento selecionado.

A **Tabela 1** lista todos os componentes que fazem parte do Conference Assistant quando o usuário o utiliza em uma conferência.

Componente	Função
Registro (J)	Adquirir informações de contato, interesses e colegas participantes.
Memo (J)	Adquirir anotações do usuário e informações relevantes a respeito da apresentação.
Recomendador (I)	Localizar apresentações interessantes.
Usuário (S)	Agregar todas as informações a respeito do usuário.
Questão (J)	Adquirir questões levantadas pelos participantes e informações relevantes a respeito da apresentação.
Localização (J)	Adquirir chegada e saída dos participantes.
Conteúdo (J)	Monitorar apresentações de slides e páginas Web, capturando as alterações de conteúdo.
Captura (J)	Detectar se áudio e/ou vídeo está sendo capturado.
Apresentação (S)	Fornecer todas as informações sobre a apresentação.

Tabela 1: Componentes do Conference Assistant e suas respectivas funções
(S = Servidores, J = Janelas, I = Interpretadores) [Dey et al., 1999]

3.5 DUMMBO

DUMMBO (*Dynamic Ubiquitous Mobile Meeting Board*) é uma aplicação para captura e acesso de reuniões informais através de equipamentos móveis como uma lousa eletrônica e um sistema para captura de som [Brotherton et al., 1999]. O projeto, desenvolvido na GATECH, permite a captura automática de escrita e conversação quando dois ou mais usuários se aproximam da lousa eletrônica e iniciam uma discussão, ou utilizam uma caneta para fazer anotações. Após a reunião, os participantes podem recuperar a informação capturada (tanto o áudio quanto anotações), através de uma interface Web.

O objetivo do DUMMBO é facilitar e simplificar a captura de reuniões informais, sem que haja necessidade de prática ou treinamento especial por parte dos usuários. Na **Figura 17** são mostrados os equipamentos utilizados para captura no DUMMBO.

Figura 17: À esquerda, a parte frontal do DUMMBO, sem botões, câmeras, ou telas de computador. À direita, a parte de trás do DUMMBO, onde ficam "escondidos" os equipamentos eletrônicos, como computadores e microfones.

A captura de uma reunião informal é iniciada assim que um dos participantes começar a falar (ativação por voz) ou retirar uma caneta ou apagador do suporte localizado junto à lousa (ativação por sensor). Em períodos de longo silêncio e inatividade, a captura é interrompida.

Após a captura, as informações podem ser visualizadas através de uma interface Web, mostrada na **Figura 18**. Para isso, o usuário deve especificar a data e local aproximados em que ocorreu a reunião, através de filtros localizados na parte inferior da tela. A tela irá mostrar uma linha de tempo com os dias do mês especificado, destacando os dias em que reuniões foram capturadas. Selecionando-se o dia, outra linha de tempo irá mostrar as horas em que ocorreram atividades. O usuário poderá então ter acesso à sessão de captura desejada. A tela também mostra os nomes daqueles que participaram da reunião.

Existem basicamente duas formas de navegação pelo conteúdo de uma determinada sessão. Uma delas é através do playback a partir do início da sessão. A tela vai mostrando as anotações e o áudio conforme o tempo.

Outra forma de navegação é deslizar o seletor associado à linha de tempo até encontrar a imagem ou anotação desejada. Para disparar o áudio correspondente, basta clicar num ponto da imagem ou numa anotação feita na lousa eletrônica durante a sessão de captura (handwriting).

Figura 18: Interface de acesso à informação capturada do DUMMBO.

3.6 NOTEPALS

O *NotePals* é uma aplicação para captura de anotações através de dispositivos como PDAs (*Personal Digital Assistant*) e posterior acesso através de uma interface Web, permitindo que todos os usuários tenham acesso compartilhado às anotações. A principal diferença desta aplicação em relação aos demais sistemas de computação ubíqua é que não há necessidade de se ter um ambiente instrumentado para captura [Davis et al., 1999]. No entanto, isto implica em algumas restrições como, por exemplo, não haver possibilidade de captura de áudio e vídeo.

Os usuários do NotePals podem utilizá-lo em qualquer local para registrar suas anotações, juntamente com alguns atributos como autor, projeto e data, para que o conteúdo possa mais tarde ser enviado para um repositório compartilhado na Web e acessado por diversos outros usuários.

Na **Figura 19** é mostrada a interface do *Note Browser*, pela qual os usuários têm acesso às anotações capturadas através do NotePals. Filtros disponíveis na parte superior da tela permitem que as informações sejam recuperadas de acordo com o projeto, autor, data, tipo de anotação e palavra-chave.

Figura 19: Tela do Note Browser.

O principal benefício desta aplicação é que os usuários (alunos, grupos de pesquisa, participantes de reuniões e conferências), ao verificarem as anotações feitas por outras pessoas, terão acesso a diferentes pontos de vista sobre o assunto em discussão.

3.7 CONSIDERAÇÕES FINAIS

Vimos nesse capítulo diversos exemplos de mecanismos de visualização de informação em ambientes que utilizam captura e acesso. Através desses mecanismos pudemos observar características em comum, como a visualização de elementos de informação de acordo com o tempo, e a sincronização entre as diversas mídias em aplicações colaborativas. Outra característica comum nesses ambientes é que as informações são apresentadas a curto prazo. O próximo capítulo irá apresentar as linguagens utilizadas pela infra-estrutura de visualização de informação evolucionária implementada neste projeto.

Capítulo 4 – Padrões XML

4.1 CONSIDERAÇÕES INICIAIS

Considerados no contexto da World Wide Web, documentos hipermídia representam hoje uma ampla fonte de informação e, portanto, podem ser utilizados por aplicações que visam recuperar e apresentar grandes quantidades de dados na Web. Entretanto, pode-se observar que a estrutura de documentos largamente utilizados como o HTML (*HyperText Markup Language*) são desprovidos de semântica associada às informações.

A linguagem XML⁷ (*Extensible Markup Language*) foi definida pelo World Wide Web Consortium (W3C) como uma linguagem apropriada à representação de dados, documentos e demais entidades, cuja essência fundamenta-se na capacidade de agregar informações [Pimentel & Teixeira, 2000]. Um documento XML representa, basicamente, duas dimensões de informação: o conteúdo propriamente dito e a estrutura organizacional desse conteúdo. O padrão XML fornece recursos para a definição de classes de elementos e atributos com regras bem definidas. Outros padrões da mesma família manipulam outras dimensões de documentos, como apresentação, semântica e metadados. Um exemplo de padrão definido como XML é o padrão SVG⁸ (*Scalable Vector Graphics*), que está se tomando uma alternativa para apresentação dinâmica e interativa de informação através de gráficos vetoriais na Web.

Considerando os objetivos descritos no Capítulo 1, a infra-estrutura de visualização proposta neste trabalho visa permitir a criação dinâmica de uma representação gráfica e interativa de informações contidas em documentos XML a ser disponibilizada na Web. O padrão SVG foi explorado e utilizado como mecanismo de visualização por ser plenamente compatível com os padrões XML. Este capítulo apresenta os conceitos básicos dos padrões XML utilizados na implementação da infra-estrutura proposta neste trabalho.

⁷ World Wide Web Consortium (<http://www.w3.org/xml>)

⁸ World Wide Web Consortium (<http://www.w3.org/svg>)

4.2 XML

XML (*Extensible Markup Language*) é uma linguagem de marcação apropriada à representação de dados, documentos e demais entidades (software, relatório, protocolo, *script*, folha de estilo, etc.), cuja essência fundamenta-se na capacidade de agregar informações [Pimentel & Teixeira, 2000].

A linguagem XML permite que autores de documentos criem seus próprios conjuntos de elementos para melhor caracterizarem os documentos de seus interesses. Por conseguinte, aplicações podem associar semântica a dados e campos de documentos, além de viabilizar o processamento automático de documentos por essas aplicações. XML permite descrever dados estruturados de forma extensível e independente de plataforma. O padrão XML é recomendação da World Wide Web Consortium desde fevereiro de 1998 e é suportado pelos principais *browsers* do mercado.

Um documento XML é formado basicamente por elementos hierarquicamente estruturados e seus respectivos atributos. Um elemento é utilizado de acordo com a semântica a ele atribuída pelo desenvolvedor da aplicação. Um elemento pode conter outros elementos e atributos (sendo classificado como elemento do tipo complexo) ou não (sendo classificado como elemento do tipo simples). Já um atributo é uma propriedade relacionada ao elemento e não pode conter outros elementos ou atributos.

Documentos XML que seguem corretamente as regras sintáticas para sua composição são ditos documentos **bem-formados**. Por exemplo, todo elemento em XML deve conter uma *tag*⁹ de abertura e uma *tag* de fechamento e todo documento XML deve possuir um único elemento raiz. O exemplo a seguir é uma instância de um documento XML, onde o elemento *disciplina* contém os atributos *nome* e *semestre*, é do tipo complexo e também é o elemento raiz do documento. Os elementos *docente* e *aluno* são do tipo complexo e não contêm atributos. Os demais elementos são do tipo simples e contêm valores associados.

⁹As *tags* XML são símbolos representados por "<" e ">" que indicam que o conteúdo entre esses símbolos é um elemento


```

<?xml version="1.0" encoding="UTF-8"?>
<disciplina nome="Hipermídia" semestre="2 2002">
  <docente>
    <nome>Teresa</nome>
    <departamento>ICMC</departamento>
  </docente>
  <aluno>
    <nome>Aline</nome>
    <email>aline@com</email>
  </aluno>
  <aluno>
    <nome>Gustavo</nome>
    <email>gustavo@com</email>
  </aluno>
  <aluno>
    <nome>Marcelo</nome>
    <email>marcelo@com</email>
  </aluno>
  <aluno>
    <nome>Roberto</nome>
    <email>roberto@com</email>
  </aluno>
  <aluno>
    <nome>Silvana</nome>
    <email>silvana@com</email>
  </aluno>
</disciplina>

```

Os documentos XML **válidos** são documentos bem-formados que seguem as especificações contidas em seus respectivos XML Schemas¹⁰, conforme é apresentado na próxima seção.

4.3 XML SCHEMA

Para a especificação de documentos XML, é importante determinar a sequência e o encadeamento dos elementos que os compõem, bem como atributos, valores e tipos de dados. Tal especificação é feita na forma de gramáticas para documentos XML e pode ser feita através de DTDs (Document Type Definitions)¹¹ ou através de XSDs (XML Schema Definition). O XML Schema possui como princípios básicos:

- ❑ Definir quais elementos e atributos podem aparecer num documento XML;
- ❑ Definir se os elementos são do tipo simples ou complexo;
- ❑ Definir a ordem em que os elementos devem aparecer no documento;
- ❑ Definir os tipos de dados que podem estar contidos nos elementos e atributos.

¹⁰ World Wide Web Consortium (<http://www.w3.org/XML/Schema>)

¹¹ World Wide Web Consortium (<http://www.w3.org/XML/>)

Considerando o exemplo de instância XML apresentada na seção anterior, o trecho a seguir corresponde a um XML Schema que pode ser utilizado para especificar aquela instância.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="disciplina">
    <xsd:complexType>
      <xsd:attribute name="nome" type="xs:string"/>
      <xsd:attribute name="semestre" type="xs:string"/>
      <xsd:sequence>
        <xsd:element name="docente" type="docenteType" maxOccurs="10"/>
        <xsd:element name="aluno" type="alunoType" maxOccurs="100"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="docenteType">
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"/>
      <xsd:element name="departamento" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="alunoType">
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"/>
      <xsd:element name="email" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

De acordo com esse XML Schema, os atributos *nome* e *semestre* do elemento *disciplina* devem conter valores formados por cadeias de caracteres (*string*). O elemento *docente* deve aparecer sempre antes do elemento *aluno* e possui os elementos filhos *nome* e *departamento*. Já o elemento *aluno* possui os elementos filhos *nome* e *email*. Neste exemplo, os valores de mínima (*minOccurs*) e máxima (*maxOccurs*) ocorrência de alguns elementos foram atribuídos, caso contrário os valores tomados como *default* são zero e um, respectivamente.

4.4 SVG

SVG (*Scalable Vector Graphics*) é uma linguagem de marcação para especificação de gráficos vetoriais, definida sob coordenação do W3C¹². Os documentos correspondentes podem ser apresentados de forma isolada ou embutidos em arquivos HTML e XML. Através das especificações do SVG, é possível criar diversos tipos de gráficos bidimensionais, como linhas, círculos e polígonos, bem como incluir preenchimento usando gradiente de cores, controle de opacidade, transformações, entre outros.

Diversos exemplos de aplicações que utilizam SVG podem ser encontrados na Web. Southard¹³, por exemplo, propõe um algoritmo para geração de árvores SVG correspondentes à estrutura de documentos HTML e XML. A metáfora de árvore utiliza galhos para representar os elementos que compõem documentos e folhas para representar os atributos de cada elemento, como pode ser observado na **Figura 20**. Esse algoritmo proporciona interatividade durante a visualização das árvores geradas: quando o mouse é posicionado sobre galhos ou folhas, o nome do elemento ou o conteúdo de um atributo é apresentado ao usuário.

Figura 20: Árvore SVG implementada por Southard¹³. Ao posicionar o mouse sobre uma folha, o nome do atributo correspondente e seu conteúdo são mostrados.

¹² World Wide Web Consortium (<http://www.w3.org/svg>)

¹³ XML Groove (<http://www.jeffsouthard.com/demos/grove>)

O principal propósito do algoritmo de Southard, segundo o autor, é familiarizar desenvolvedores e projetistas com o conceito de árvore como estrutura do XML e também com SVG.

A partir da implementação da Árvore SVG de Southard, foi desenvolvido no contexto deste trabalho uma proposta de extensão desta árvore para que ela refletisse não a estrutura de um documento XML, mas sim, a estrutura de um *website*. Assim, cada galho da árvore representa uma página web (elemento) e as informações correspondentes a esta página são mostradas no canto direito da tela (atributos). O resultado final deste trabalho é a *Winter Tree*, a qual pode ser observada na **Figura 21**.

Figura 21: Winter Tree associada a um site de autoria colaborativa.

Outras aplicações como o *AgileBlox* da *Elansoft*¹⁴ permitem que o usuário gere automaticamente gráficos SVG em forma de barras ou “pizzas”, a partir de dados que o usuário provê ao preencher um formulário. A **Figura 22** é um exemplo de gráfico em forma de “pizza” que pode ser gerado através de uma página de demonstração disponível no *site* da Elansoft.

¹⁴ Elansoft (<http://www.elansoft.com>)

Figura 22: Exemplo de um *piechart* gerado no AgileBlox da Elansoft.

Outras aplicações que utilizam SVG podem ser encontradas na página da Adobe¹⁵ como, por exemplo, gráficos interativos, imagens que simulam uma aparência tridimensional, visualização interativa de prédios e teatros e geração dinâmica de documentos SVG através da integração de outras tecnologias e padrões.

4.5 XSLT

Documentos XML, em sua forma básica, não declaram informações relacionadas à apresentação. Para que documentos XML possam ser apresentados, utilizam-se especificações padronizadas. Dentre as especificações existentes, a linguagem XSL (*Extensible Stylesheet Language*) foi proposta pelo W3C para declarar folhas de estilo (<http://www.w3.org/Style/XSL>). Através da linguagem XSL, um projetista pode definir quais elementos e atributos do documento XML serão apresentados, assim como sua ordem de apresentação e formato (texto, gráfico, entre outros).

A linguagem XSL possui diversos componentes, entre eles o XSLT (XSL Transformation), responsável pelo fornecimento de informações sobre o processamento e a apresentação de documentos XML. Em geral, XSLT é utilizado para transformar documentos XML em outros documentos XML de estrutura mais simples, ou em outros tipos de documentos reconhecíveis por meio do *browser*, através da associação de regras de estilo a elementos

¹⁵ SVG Zone (<http://www.adobe.com/svg/demos/main.html>)

da estrutura hierárquica de documentos XML. Sendo assim, XSLT permite transformar documentos XML em documentos SVG.

Suponha que se queira apresentar a instância XML da Seção 4.2 através de uma representação SVG. O trecho a seguir contém instruções do documento XSLT responsável por tal transformação. Cada conjunto de instrução do XSLT é definido através de *templates*. O primeiro *template* irá definir as dimensões da janela de exibição da representação SVG. O segundo *template* irá acessar o conteúdo dos elementos *docente* e *aluno*, e mostrar esse conteúdo na forma de texto.

```
<?xml version="1.0" encoding="UTF 8"?>
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <svg width="400" height="200"> <!-- dimensões da janela de exibição -->
      <xsl:apply-templates/>
    </svg>
  </xsl:template>

  <xsl:template match="disciplina">
    <text font-family="Verdana" font-size="20" fill="black" x="10"
      y="30"><xsl:value-of select="@nome"/> - <xsl:value-of select="@semestre"/>
    </text>
    <xsl:for-each select="docente">
      <xsl:variable name="y" select="count(preceding-sibling::*)*15 + 60"/>
      <xsl:variable name="y2" select="$y+5"/>
      <circle cx="20" cy="{ $y}" r="5" stroke-width="1" stroke="black"
        fill="#CCCCCC"/>
      <text font-family="Verdana" font-size="16" fill="black" x="35" y="{ $y2}">
        <xsl:value-of select="nome"/> - <xsl:value-of select="departamento"/>
      </text>
    </xsl:for-each>
    <xsl:for-each select="aluno">
      <xsl:variable name="y" select="count(preceding-sibling::*)*15 + 75"/>
      <xsl:variable name="y2" select="$y+5"/>
      <circle cx="25" cy="{ $y}" r="4" stroke-width="1" stroke="black"
        fill="black"/>
      <text font-family="Verdana" font-size="12" fill="black" x="40" y="{ $y2}">
        <xsl:value-of select="nome"/> - <xsl:value-of select="email"/>
      </text>
    </xsl:for-each>
  </xsl:template>
```

</xsl:stylesheet>

Documentos XSLT são instâncias de documentos XML formadas por um grupo de elementos com funções específicas, mas que possuem a mesma sintaxe do documento XML e, portanto, podem ser analisados da mesma forma que um simples documento XML. Basicamente, o que difere a instância de um documento XML de uma instância de documento XSLT é a existência de um processador XSLT que tem a capacidade de interpretar os elementos do documento XSLT e executar funções distintas para cada um deles. O processador faz uso das instruções contidas na folha de estilo, determinando como o novo documento deve ser gerado. Na próxima seção são apresentados alguns exemplos de processadores XSLT.

4.6 PROCESSADORES XSLT

Aplicações que utilizam SVG podem ser utilizadas para apresentar dados provenientes de fontes externas como, por exemplo, documentos XML. Nesse caso, esses documentos podem servir como base de informações para extração, através de algoritmos contidos em documentos XSLT, das informações a serem apresentadas. Esses algoritmos definem também a forma como essas informações são apresentadas ao serem transformadas em SVG. Processadores como o XT¹⁶ e o Saxon¹⁷ podem ser utilizados nessa transformação.

Neste projeto utilizou-se o Saxon, ilustrado na **Figura 23**, desenvolvido por Michael Kay e que pode ser obtido gratuitamente na Web. O Saxon é um pacote de programas para processamento de documentos XML implementado sob a plataforma Java. Esse pacote inclui um processador de documentos XSLT, uma biblioteca de funções escritas em Java e um analisador léxico de documentos XML.

¹⁶ XT (<http://www.blnz.com/xt/index.html>)

¹⁷ Saxon (<http://saxon.sourceforge.net/>)

Figura 23: Com o Saxon, utilizando-se documentos XML e XSLT (pode-se também combinar vários documentos XSLT) como entrada, é possível gerar outros tipos de documentos como HTML, SVG, JavaScript, ou ainda outros documentos XML e XSLT.

A **Figura 24** é um exemplo do resultado do processamento, utilizando-se o Saxon, da instância XML apresentada na Seção 4.2 com o documento XSLT apresentado na Sessão 4.5, produzindo como saída uma representação SVG, que pode ser visualizada usando um *plug-in* para *browser*:

Figura 24: Informações exibidas através de uma representação SVG utilizando *plug-in* para *browser*

4.6 CONSIDERAÇÕES FINAIS

Apesar dos diversos recursos providos pelo padrão SVG, documentos especificados nesse padrão necessitam de *plug-in* específico para serem apresentados na Web ou processadores SVG para a visualização de documentos isolados. Entretanto, o uso de SVG é facilitado pelo fato de existir uma série de ferramentas para visualização de SVG gratuitas que podem

ser encontrados na Web. Por exemplo, o *SVG Viewer* da Adobe¹⁸ é um dos visualizadores SVG mais utilizados atualmente e pode ser facilmente instalado em *browsers* para Web.

SVG é uma linguagem relativamente nova, sendo recomendação do W3C desde setembro de 2001. No entanto, diversas aplicações vêm surgindo na Web, conforme visto na Seção 4.4. Muitas dessas aplicações procuram explorar características do SVG como a capacidade de integração com diversos padrões XML, o seu formato texto, o seu conteúdo extensível e a possibilidade de ser gerado dinamicamente. No próximo capítulo, é apresentada uma infra-estrutura que utiliza os padrões XML para prover visualização de informação evolucionária.

¹⁸ SVG Viewer (<http://www.adobe.com/svg/viewer/install/main.html>)

Capítulo 5 – Infra-estrutura iVIEW

5.1 CONSIDERAÇÕES INICIAIS

Este trabalho tem como objetivo investigar mecanismos para a visualização de informação evolucionária no contexto da World Wide Web, o que levou à definição da infra-estrutura iVIEW – infra-estrutura de **V**isualização de **I**nformação **E**volucionária na **W**eb.

Durante o processo de implementação da infra-estrutura iVIEW, os padrões XML descritos no capítulo anterior foram utilizados para:

- Definir a estrutura dos documentos de entrada que contêm as informações a serem visualizadas (XML Schema);
- Construir algoritmos que executam o processamento necessário para a obtenção de uma representação gráfica das informações de entrada (XSLT);
- Prover a representação gráfica das informações a serem visualizadas (SVG).

O público alvo da infra-estrutura iVIEW são *desenvolvedores de aplicações de captura e acesso* (referenciados apenas como *desenvolvedores* no restante desta dissertação) que necessitam apresentar uma grande quantidade de informação capturada a longo prazo, em particular quando as características de informação evolucionária – expansão progressiva e referência intrínseca – devem ser visualizadas na Web.

Neste capítulo são descritos os elementos de informação e as estruturas de visualização do iVIEW, relacionando-os com os tipos de dados apresentados nos Capítulos 2 e 3 que são suportados pela infra-estrutura. A estrutura dos documentos de entrada e os recursos de interação com o usuário também são apresentados, juntamente com as etapas necessárias para a obtenção de uma representação gráfica de informações utilizando SVG.

5.2 VISÃO GERAL DA INFRA-ESTRUTURA IVIEW

A infra-estrutura iVIEW tem como objetivo primordial prover a visualização de informação evolucionária extraída de documentos XML utilizando representações gráficas em SVG.

Para que essa representação gráfica seja obtida, foram definidos:

- Um conjunto de documentos necessários para o processo de transformação de informações;
- Um conjunto de etapas de processamento desses documentos.

A **Figura 25** é um esquema resumido dos documentos, recursos e etapas de processamento de documentos que compõem a infra-estrutura iVIEW. Nessa infra-estrutura:

- O desenvolvedor é responsável por preparar o Documento 2 (layout.xml), que define a estrutura de apresentação das informações e instâncias do Documento 1 (aplicacao.xml), que é documento utilizado como fonte das informações que serão visualizadas.
- Os Documentos 3 (layout.xsd), 4 (template.xslt) e 7 (templateJS.xslt) foram elaborados no contexto deste trabalho e fazem parte da infra-estrutura.
- Os Documentos 5 (aplicacao.xslt), 6 (aplicacao.svg) e 8 (aplicacao.js) são gerados automaticamente após cada etapa de processamento.

O processo de geração de visualização de informações em SVG é composto por três etapas:

- Etapa 1: Geração do Documento de Transformação (Entrada: Documentos 2 e 4. Saída: Documento 5);
- Etapa 2: Geração da Representação SVG (Entrada: Documentos 1 e 5. Saída: Documento 6);
- Etapa 3: Geração do Documento com Funções de Interação (Entrada: Documentos 2 e 7. Saída: Documento 8).

Figura 25: Etapas para obtenção de uma visualização de informações em SVG. Os Documentos 1 e 2 são criados pelo desenvolvedor. O restante dos documentos são providos pela infra-estrutura iVIEW (3, 4 e 7) ou gerados automaticamente (5, 6 e 8). As linhas duplas que unem o Documento 2 ao Documento 3 significa que o Documento 2 deve obedecer as especificações contidas no Documento 3.

Os documentos numerados na **Figura 25** são os documentos de entrada e de saída processados durante cada etapa:

- Documento 1 (aplicacao.xml): é um documento XML contendo as informações que os usuários irão visualizar e com as quais irão interagir. É o primeiro documento que o desenvolvedor da aplicação deve criar para iniciar o processo de obtenção de uma visualização de informação em SVG. Este documento será referenciado, no restante desta dissertação, como documento XML fonte.
- Documento 2 (layout.xml): é um documento XML contendo a especificação da estrutura de visualização e a definição dos elementos a serem apresentados numa determinada aplicação, o qual o desenvolvedor deverá criar tendo como base o documento XML fonte (aplicacao.xml);
- Documento 3 (layout.xsd): é um XML Schema que define as regras de formação e estruturação do Documento 2 (layout.xml). Esse documento faz parte da infra-estrutura iVIEW e deve ser utilizado pelo desenvolvedor para saber como gerar o Documento 2 (layout.xml);

- ❑ Documento 4 (template.xslt): é um documento XSLT genérico, que também faz parte da infra-estrutura iVIEW, contendo instruções para geração de outro documento XSLT, que é específico para a aplicação definida pelo desenvolvedor. Para a geração deste documento específico, o documento XSLT genérico toma como entrada as informações contidas no Documento 2 (layout.xml);
- ❑ Documento 5 (aplicacao.xslt): é um documento XSLT gerado automaticamente contendo instruções que irão gerar uma representação SVG do Documento 1 (aplicacao.xml);
- ❑ Documento 6 (aplicacao.svg): é um documento SVG, gerado automaticamente, que apresenta graficamente instâncias do Documento 1 (aplicacao.xml), da forma como definida no Documento 2 (layout.xml);
- ❑ Documento 7 (templateJS.xslt): é um documento XSLT genérico que faz parte da infra-estrutura iVIEW e gera um documento JavaScript específico para a aplicação do desenvolvedor, e toma como entrada as informações contidas no Documento 2 (layout.xml);
- ❑ Documento 8 (aplicacao.js): é um documento JavaScript específico, gerado automaticamente, que contém funções que provêem a interação dos usuários com o Documento 6 (aplicacao.svg).

5.3 ELEMENTOS E ESTRUTURAS DE VISUALIZAÇÃO DA IVIEW

5.3.1 Estrutura Básica de Visualização

Visando dar suporte à apresentação de informações provenientes de documentos XML e oferecer recursos de interação com essas informações, foi estabelecido que a estrutura básica de visualização do iVIEW seria constituída por duas áreas principais:

- ❑ Uma área para a exibição de grupos de elementos de visualização com suas características e atributos;
- ❑ Uma área contendo filtros de seleção para mostrar ou ocultar determinados grupos de elementos.

O posicionamento e as dimensões dessas áreas são definidos pelo desenvolvedor que fará uso do iVIEW para uma determinada aplicação. Nas **Figuras 26 e 27** são apresentados dois exemplos de como a estrutura básica de visualização pode ser configurada:

Figura 26: Exemplo de configuração da estrutura básica de visualização do iVIEW – disposição vertical das áreas principais.

Figura 27: Exemplo de configuração da estrutura básica de visualização do iVIEW – disposição horizontal das áreas principais.

5.3.2 Grupos de Elementos de Visualização

Os grupos de elementos de visualização que compõem uma das áreas principais do iVIEW são compostos por elementos XML de mesma natureza, que contêm os mesmos atributos e que executam as mesmas ações como resposta à interação dos usuários (as funções de

interação serão descritas na Seção 5.4.3). Por exemplo, o corpo docente de uma determinada instituição de ensino pode ser visto como um grupo, onde cada docente está associado a um elemento XML. Supondo que todos os elementos possuam os atributos “nome”, “departamento” e “ano de ingresso”, pode-se dizer que eles têm a mesma natureza, as mesmas características.

Na **Figura 28** é apresentada uma possível visualização de três grupos de elementos XML distintos:

Figura 28: Grupos de elementos XML distintos.

Como ilustrado na **Figura 28**, os elementos são o foco de visualização dos usuários e podem ser representados graficamente por formas SVG, como barras, círculos, retângulos, entre outros. Cada elemento representa um item de informação, o qual contém um ou mais atributos que podem estar relacionados a outros elementos.

De modo geral, esses grupos de elementos podem ser associados a:

- ❑ Dados temporais (Seção 2.4), por exemplo, datas de aulas capturadas;
- ❑ Dados unidimensionais (Seção 2.3.1), por exemplo, URLs;
- ❑ Dados multi-dimensionais (Seção 2.3.4), por exemplo, pessoas (alunos e docentes);

No Capítulo 6 é apresentado um exemplo no qual grupos de elementos são associados a dados provindos de ambientes de captura, conforme visto no Capítulo 3.

Outro aspecto relevante é o fato de que todos os elementos podem estar relacionados entre si, sob algum aspecto. Esse relacionamento pode ser estabelecido e visualizado através de filtros, descritos na próxima seção, e funções de interação, descritos na Seção 5.4.3.

Por permitir a apresentação de conjuntos de elementos, os Grupos de Elementos de Visualização permitem a representação da característica de expansão progressiva de informações evolucionárias. A característica de referência intrínseca é visualizada através de filtros e funções de interação apresentados na próxima seção e na Seção 5.4.3, respectivamente.

5.3.3 Filtros

Eventualmente, a quantidade de itens de informação apresentados ao usuário pode ser grande e há casos em que o usuário pode estar interessado em visualizar apenas os elementos com atributos que contenham um determinado valor. Os filtros de seleção de elementos foram implementados para prover ao usuário a opção de visualizar apenas determinados elementos – nesse caso atributos comuns permitem que sejam visualizadas referências intrínsecas entre elementos.

Pode-se considerar os grupos de elementos de visualização como listas organizadas sob um determinado aspecto. Assim, a utilização e a combinação de filtros de seleção sobre esses grupos de elementos podem criar o efeito de união ou intersecção de listas.

Na **Figura 29** pode-se observar um exemplo de utilização de um filtro de seleção. Na **Figura 29.a** é mostrado um grupo de alunos e um filtro que seleciona estes alunos de acordo com o curso do qual fazem parte. Na **Figura 29.b** o filtro é utilizado, e somente alunos dos cursos “Computação” e “Física” são exibidos.

Figura 29: Filtro de seleção de elementos. a) Todos os alunos de todos os cursos são exibidos. b) Apenas os alunos de "Computação" e "Física" são exibidos.

5.4 ETAPAS PARA GERAÇÃO DA REPRESENTAÇÃO GRÁFICA DE INFORMAÇÕES EM SVG

Na seção anterior foram apresentados os componentes básicos que fazem parte da visualização gerada pelo iVIEW e como estes podem ser organizados quanto à forma de apresentação. Nesta seção, cada uma das três etapas que compõem o processo de geração de representações gráficas utilizando SVG no iVIEW citadas na Seção 5.2 é detalhada, assim como a estrutura e o conteúdo dos documentos que fazem parte desse processo.

5.4.1 Etapa 1: Geração do Documento de Transformação

O objetivo da primeira etapa para a obtenção de uma representação gráfica e interativa de um conjunto de informações é a **geração de um documento XSLT específico** contendo instruções que irão gerar a representação SVG dessas informações a partir de um documento `aplicacao.xml` contendo os dados a serem visualizados.

Conforme mencionado no Capítulo 4, um processador XSLT como o Saxon, por exemplo, é utilizado para a geração desse documento. Como entrada, o desenvolvedor deve elaborar um documento XML (Documento 2 da **Figura 25**, `layout.xml`) contendo: (a) a

especificação do layout a ser exibido, (b) a forma de apresentação dos elementos de informação que estarão presentes na visualização e (c) alguns parâmetros necessários para estabelecer o relacionamento entre os elementos. A especificação desse documento XML foi definida durante a implementação da iVIEW através de um XML Schema, o `layout.xsd`, que se encontra na íntegra no **Apêndice A** desta dissertação. Deste modo, o desenvolvedor deverá criar uma instância XML para definir a forma de apresentação de sua aplicação, cumprindo as regras de formação e estruturação contidas no XML Schema `layout.xsd`.

Basicamente, o documento `layout.xsd` define as três partes principais que estabelecem o aspecto e as propriedades da visualização final:

- O **layout**, que torna possível a configuração de dimensões, posicionamento e cores dos objetos apresentados;
- Os **elementos**, que contêm informações de como os elementos de visualização são mostrados e que funções de interação podem atuar sobre eles;
- Os **filtros**, que contêm informações sobre a aparência e comportamento dos filtros utilizados.

O trecho de um documento `layout.xml` (Documento 2 da **Figura 25**), apresentado a seguir, define o layout da visualização final de um documento XML fonte (`aplicacao.xml`) que contém dados sobre sessões capturadas em salas de aula, referências bibliográficas, alunos e docentes de uma determinada instituição:

```
<layout>
  <janela largura="770" altura="470" cor_de_fundo="#648EB0"/>
  <titulo label="Disciplinas 1o.Sem 2001 - 2o.Sem 2002"
fonte_face="verdana" fonte_tamanho="16" fonte_cor="black"/>
  <script arquivo="disciplinas.js"/>
  <barras>
    <barra nome="sessoes_capturadas" espessura="3" altura="30"/>
    <barra nome="referencias" espessura="4" altura="32"/>
  </barras>
  <circulos>
    <circulo nome="docentes" raio="4" stroke_espessura="1"/>
    <circulo nome="alunos" raio="4" stroke_espessura="1"/>
  </circulos>
</layout>
```

Neste exemplo, os atributos do elemento *janela* definem a largura e altura máximas da área de visualização final, bem como sua cor de fundo. O elemento *script* contém o atributo *arquivo* que define qual documento JavaScript será utilizado para prover interação do usuário com a visualização final (o documento JavaScript *aplicacao.js* será gerado automaticamente na terceira e última etapa). Por fim, os elementos *barra* e *circulo* definem como os elementos de visualização serão representados: sessões capturadas e referências bibliográficas serão representadas por barras; alunos e docentes serão representados por círculos. Eventualmente, essa representação poderia ser qualquer outra forma básica do SVG, como retângulos, por exemplo.

Nesse mesmo documento *layout.xml*, a parte de definição de elementos estabelece a forma de apresentação e as propriedades dos elementos que serão visualizados:

```
<elementos>
<elemento caminho="disciplinas/bibliografia" grupoid="bibl">
  <area x="10" y="320" largura="545" altura="40" stroke_espessura="2"
stroke_cor="white" cor_de_fundo="#C6D5E1"/>
  <titulo nome="referencias bibliograficas" x="10" y="315"
fonte_cor="white" fonte_tamanho="11"/>
  <representacao nome="referencias" stroke_cor="black" cor="white"/>
  <posicao_inicial min="1" max="50" x="18" y="324"/>
  <deslocamento>10</deslocamento>
  <atributos>
    <atributo nome="titulo"/>
    <atributo nome="autor"/>
    <atributo nome="curso"/>
  </atributos>
</elemento>
<elemento caminho="disciplinas/alunos" grupoid="alns">
</elemento>
</elementos>
```

O elemento *elemento* diz respeito ao grupo de elementos de informação presentes no documento XML fonte *aplicacao.xml* – por exemplo, sessões capturadas, referências bibliográficas, docentes e alunos – e contém a definição de como esses elementos serão apresentados. O atributo *caminho* indica a localização hierárquica dos elementos dentro do documento XML fonte *aplicacao.xml* que, neste exemplo, são os elementos contendo dados de referências bibliográficas. O atributo *atributoid* contém um identificador para este

grupo de elementos, que será utilizado posteriormente por funções de interação com usuários. O elemento *area* define as dimensões e propriedades do espaço ocupado por esses elementos na visualização final. O elemento *representacao* associa elementos à sua representação SVG definida previamente na parte de layout (no caso das referências bibliográficas, foi definido que estas serão representadas por uma barra de nome “referencias”). Os elementos *posicao_inicial* e *deslocamento* indicam, respectivamente, onde o primeiro elemento deverá aparecer e o espaço em pixels entre cada elemento. Por fim, os atributos que serão visualizados são definidos pelo elemento *atributos*.

A **Figura 30** é uma representação SVG que ilustra um grupo de elementos em forma de barra:

Figura 30: Apresentação de elementos correspondentes a referências bibliográficas, conforme definidos na instância XML que especifica o formato da visualização final.

O trecho a seguir é a terceira e última parte do documento *layout.xml* que está sendo apresentado como exemplo nesta seção, no qual são definidas as propriedades dos filtros de seleção de elementos:

```
<filtros presente="sim">
  <título label="Filtros" fonte_tamanho="14" fonte_cor="blue" x="645"
y="25"/>
  <area x="565" y="10" largura="195" altura="450"
stroke_espessura="2" stroke_cor="white" cor_de_fundo="#98B4CB"/>
  <filtro nome="referencias" descricao="Ocultar/mostrar referencias:"
fonte_cor="blue" fonte_tamanho="13" x="550" y="0">
    <grupos elem grupoid="bibl" atributoid="2"/>
    <item label="Hipermidia" cor="black" tamanho="12"/>
    <item label="Multimidia" cor="black" tamanho="12"/>
    <item label="HCI" cor="black" tamanho="12"/>
  </filtro>
</filtros>
```

O atributo *presente* do elemento *filtros* indica a presença de filtros na visualização final, pois eles podem não estar presentes, se o desenvolvedor assim desejar. Os elementos *título* e *area* definem, respectivamente, o título e a área ocupada pelos filtros de seleção. Em seguida, cada filtro utilizado na aplicação do desenvolvedor é definido quanto às suas propriedades. Neste exemplo, o filtro de nome “referencias” irá atuar sobre o grupo de elementos cujo atributo *atributoid* é “bibl”, ou seja, o grupo que contém dados de referências bibliográficas. A seleção é composta por itens cujos atributos *label* contêm os valores “Hiperinídia”, “Multimídia” e “HCT”. Assim, os usuários da aplicação poderão visualizar apenas as referências bibliográficas do curso “Hiperinídia”, ou dos cursos “Multimídia” e “HCT”, ou quaisquer combinações que desejar.

Para que o documento *layout.xml* anterior seja válido, é necessário seguir as definições contidas no documento *layout.xsd* (Documento 5 da **Figura 25**). O trecho a seguir, extraído do documento *layout.xsd* corresponde à definição do elemento do tipo complexo *elemento*:

```
<xsd:element name="elemento">
  <xsd:complexType>
    <xs:attribute name="caminho" type="xs:string"/>
    <xs:attribute name="grupoid" type="xs:string"/>
    <xsd:sequence>
      <xsd:element name="area" type="areaType" minOccurs="1"/>
      <xsd:element name="titulo" type="tituloType" minOccurs="1"
        maxOccurs="1"/>
      <xsd:element name="comandos" type="comandosType" minOccurs="1"/>
      <xsd:element name="representacao" type="representacaoType" minOccurs="1"/>
      <xsd:element name="posicao_inicial" type="posicao_inicialType"
        minOccurs="1" maxOccurs="50"/>
      <xsd:element name="deslocamento" type="xsd:string" minOccurs="1"/>
      <xsd:element name="atributos" type="atributosType" minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name="atributosType">
  <xsd:sequence>
    <xsd:element name="atributo" type="atributoType" minOccurs="1"
      maxOccurs="6"/>
  </xsd:sequence>
</xsd:complexType>
```

Este XML Schema `layout.xsd` define a sequência e a quantidade possível de ocorrências de cada elemento dentro da instância XML correspondente, bem como o tipo de dados que cada elemento e atributo pode conter. Por exemplo, no trecho apresentado, o elemento *elemento* possui os atributos *caminho* e *grupoid* e uma sequência dos seguintes elementos: *area*, *titulo*, *comandos*, *representacao*, *posicao_inicial*, *deslocamento* e *atributos*, cada um com sua cardinalidade (*minOccurs* e *maxOccurs*).

Opcionalmente, o desenvolvedor pode criar um documento `layout.xml` automaticamente através de uma interface escrita em PHP¹⁹, que também faz parte da infra-estrutura iVIEW. Na **Figura 31** é apresentado o formulário que o desenvolvedor pode preencher e utilizar para gerar o documento `layout.xml`.

Layout

Área Total

Largura

770

Altura

470

Cor de fundo

#E2ECF5

Título

Font-face

verdana

Font-size

16

Font-color

#222222

Arquivo Javascript

.js

Cores de Gradiente

Gradiente 1

Nome

Gradiente1

x1

x2

y1

y2

Stop 1

Offset

%

Stop-color

Stop 2

Offset

%

Stop-color

Barras

Barra 1

Nome

Espessura

Altura

Barra 2

Nome

Espessura

Altura

Círculos

Círculo 1

Nome

Espessura do Stroke

1

Ralo

4

Retângulos

Retângulo 1

Nome

Espessura do Stroke

1

Largura

8

Altura

8

Caixa de Exibição de Atributos

Número máximo de caracteres exibidos

40

Font-face

arial

Font-size

10

Font-color

red

¹⁹ (<http://www.php.net>)

Elementos

Elemento 1

Caminho

Groupid

Área Ocupada

Posição x

Posição y

Largura

Altura

Espessura do Stroke

Cor do Stroke

Cor de Fundo

Nome do grupo de elementos

Posição x

Posição y

Font-size

Font-color

Representação SVG

Nome

Cor do Stroke

Cor de Preenchimento

Deslocamento entre elementos

Fileira 1

Posição inicial

x

y

Do elemento

ao

Atributos que serão exibidos

Atributo 1

Famandos da Interação

Mouseover

mouseoverObject(evt,1)

Mouseout

mouseoutObject(evt,1)

Mousedown ativo

sim

Atributo comparado

Nova cor de stroke

Nova cor de preenchimento

Grupo de elementos relacionados

Grupo 1

Groupid

Atributoid (número do atributo que será comparado)

Cor de stroke original

Nova cor de stroke

Cor de preenchimento original

Nova cor de preenchimento

Filtros

Nome do filtro

Posição x

Posição y

Font-size

Font-color

Área Ocupada

Posição x

Posição y

Largura

Altura

Espessura do Stroke

Cor do Stroke

Cor de Fundo

Filtro 1

Nome

Descrição

Font-size

10

Font-color

Posição x

Posição y

Grupo de elementos sobre os quais irá atuar

Grupo 1

Groupid

Atributoid (número do atributo que será comparado)

Itens de seleção do filtro (os labels devem ser os valores que o atributo comparado pode assumir)

Item 1

Label

Cor da fonte

Tamanho da fonte

Enviar

Figura 31: Formulário para geração de uma instância XML para especificação de layout, elementos e filtros (Documento 2 da Figura 25).

Estando o documento layout.xml devidamente especificado, este será processado por um processador XSLT juntamente com um documento XSLT genérico – template.xslt

(Documento 4 da **Figura 25**). O documento `template.xslt` tem como função utilizar as especificações contidas no documento `layout.xml` para gerar o documento `aplicacao.xslt` que, por sua vez, irá processar o documento `aplicacao.xml`. O documento `template.xslt`, presente na íntegra no **Apêndice B**, também faz parte da infra-estrutura iVIEW.

Na próxima seção são descritos como um documento XSLT pode transformar um documento XML em SVG e a estrutura básica do documento SVG gerado como resultado do processamento.

5.4.2 Etapa 2: Geração da Representação SVG

Nesta etapa, o documento `aplicacao.xslt` gerado na etapa anterior é processado por um processador XSLT juntamente com o documento `aplicacao.xml`. Como saída, é produzido um documento SVG que apresenta graficamente as informações do documento `aplicacao.xml` no formato especificado pelo desenvolvedor na etapa anterior.

O trecho do documento `aplicacao.xslt` para a aplicação apresentada na Etapa 1, mostrado a seguir, contém instruções que transformam informações contidas no documento `aplicacao.xml` na representação gráfica SVG correspondente (`aplicacao.svg`). As linhas do código estão numeradas para facilitar a descrição.

```
(1)<xsl:template match="disciplinas/bibliografia">
(2)  <rect x="10" y="320" height="40" width="545" stroke="white" stroke-
width="2" fill="#C6D5E1"/>
(3)  <text style="font-size:11; fill:white" x="10" y="315">referencias
</text>
(4)  <g onmouseover="mouseoverObject(evt,3)" onmouseout="mouseoutObject
(evt,3)">
(5)    <xsl:variable name="n" select="count(descendant::*)"/>
(6)    <count id="bibl">
(7)      <num><xsl:value-of select="$n"/></num>
(8)    </count>
(9)    <xsl:for-each select="*">
```



```

(10) <xsl:variable name="deslocaX" select="count(preceding-
sibling::*)*10 + 18"/>
(11) <xsl:variable name="gotid" select="count(preceding-sibling::*)+1"/>
(12) <g id="bibl{$gotid}" transform="translate({$deslocaX},324)">
(13) <xsl:variable name="param" select="@cursoid"/>
(14) <use xlink:href="#book_url" fill="url(#Gradient2)"
stroke="url(#Gradient2)" onclick="showrelatedbibl(evt,'{$param}')"
visibility="visible">yes</use>
(15) <inactivefilters>0</inactivefilters>
(16) <atr><xsl:value-of select="@titulo"/></atr>
(17) <atr><xsl:value-of select="@autor"/></atr>
(18) <atr><xsl:value-of select="@cursoid"/></atr>
(19) </g>
(20) </xsl:for-each>
(21) </g>
(22)</xsl:template>

```

Um documento XSLT especifica transformações através de uma linguagem declarativa, sendo possível acessar e manipular diretamente os elementos de um documento XML através do reconhecimento de sua estrutura hierárquica. Assim, alguns cálculos como o deslocamento horizontal e vertical de elementos na visualização final podem ser feitos com base na contagem de seus elementos precedentes no documento fonte (linha 10 do trecho apresentado). As instruções XSLT também podem acessar diretamente o valor dos atributos dos elementos de visualização, utilizando o símbolo “@” antes do nome do atributo (linhas 16, 17 e 18). O conteúdo destes atributos, portanto, estará presente no documento SVG gerado e será acessado posteriormente pelas funções de interação com os usuários.

No exemplo apresentado nesta seção, as instruções XSLT geram como saída um documento no formato SVG apresentado a seguir (*aplicacao.svg*). O trecho contém apenas a representação gráfica das três primeiras referências bibliográficas do documento fonte. O atributo *mouseover*, por exemplo, indica qual função será executada ao passar o mouse sobre os elementos cujos valores do atributo *id* são “bibl1”, “bibl2”, “bibl3”. Neste exemplo, a função executada (*mouseoverObject*) irá mostrar 3 atributos relativos a cada um desses elementos. O elemento *atr* contém os valores desses atributos. O elemento *inactivefilters* contém o número de filtros ativos sobre os elementos.

```

<g onmouseover="mouseoverObject(evt,3)"
onmouseout="mouseoutObject(evt,3)">
  <count id="bibl"><num>16</num></count>
  <g id="bibl1" transform="translate(18,324)">
    <use xlink:href="#referencias" fill="white" stroke="black"
onclick="showrelatedbibl(evt,'Multimedia') "
visibility="visible">yes</use>
    <inactivefilters>0</inactivefilters>
    <atr>Mpeg 4</atr>
    <atr>R. Faria</atr>
    <atr>Multimedia</atr>
  </g>
  <g id="bibl2" transform="translate(28,324)">
    <use xlink:href="#referencias" fill="white" stroke="black"
onclick="showrelatedbibl(evt,'Hipermedia') "
visibility="visible">yes</use>
    <inactivefilters>0</inactivefilters>
    <atr>Padroes XML</atr>
    <atr>A. Siqueira</atr>
    <atr>Hipermedia</atr>
  </g>
  <g id="bibl3" transform="translate(38,324)">
    <use xlink:href="#referencias" fill="white" stroke="black"
onclick="showrelatedbibl(evt,'HCI') " visibility="visible">yes</use>
    <inactivefilters>0</inactivefilters>
    <atr>Style Sheets</atr>
    <atr>M. C. Almeida</atr>
    <atr>HCI</atr>
  </g>
</g>

```

Os trechos dos documentos apresentados até então referem-se a uma aplicação para visualização de dados sobre sessões capturadas em salas de aula, referências bibliográficas, alunos e docentes de uma determinada instituição, conforme mencionado durante a descrição das etapas 1 e 2. O documento XML fonte dessa aplicação (aplicacao.xml) é relativamente extenso, de modo que as informações nele contidas não podem ser visualizadas em uma única tela. No entanto, a visualização em uma única tela é possível através da transformação do documento XML em SVG. A **Figura 32** é a representação

Na infra-estrutura iVIEW, foram implementadas algumas funções básicas de interação dos usuários com os elementos de visualização:

- *mouseOver(evt, n)* e *mouseOut(evt, n)*: exibir o conteúdo dos *n* atributos de um dos elementos ao passar o ponteiro do mouse sobre esse elemento (o número de atributos é especificado pelo desenvolvedor). Na **Figura 33** é mostrado um exemplo.

Figura 33: As informações sobre um determinado elemento são representadas por atributos especificados pelo desenvolvedor.

- *showRelationship(evt, attributeValue)*: ao clicar com o mouse sobre um determinado elemento, sua cor é modificada, assim como a de todos os elementos que contenham um atributo com mesmo valor. O valor deste atributo é especificado como uma variável de entrada da função. Através da mesma, é possível observar o relacionamento existente entre elementos pertencentes a um mesmo grupo, ou entre elementos de grupos distintos, dependendo de como o usuário especificar. No exemplo mostrado na **Figura 34**, são exibidos dois grupos de elementos: um de orientadores e outro de alunos de pós-graduação. Ao clicar-se num determinado orientador, sua cor é automaticamente alterada, assim como a de todos os alunos sob sua orientação. Nesse caso, o valor do atributo de entrada da função é o *nome* do orientador. A definição de quais atributos devem ser comparados entre os elementos é feita no documento *layout.xml* descrito na Seção 5.3.1.

Figura 34: Exemplo de relacionamento entre elementos.

- *showFilterName(evt, attributeValue)*: através do uso de filtros de seleção, pode-se exibir ou destacar apenas os elementos de interesse para o usuário. Vale lembrar que um ou mais filtros podem atuar sobre o mesmo grupo de elementos. Um único filtro também pode atuar sobre um ou mais grupos de elementos. Na **Figura 35** são ilustrados dois exemplos da utilização de filtros. Na **Figura 35.a** o grupo de elementos representa sessões capturadas em salas de aula no decorrer de um determinado período. Dois filtros atuam sobre esse grupo, um para cursos e outro para tipo de sessão (aula, seminário ou prova). Assim, apenas as sessões do tipo “aula expositiva” dos cursos “Computação” e “Matemática” são exibidos. Na **Figura 35.b** tem-se um único filtro atuando sobre dois grupos distintos de elementos. Dessa forma, somente orientadores e alunos dos cursos “Computação” e “Matemática” são exibidos.

Figura 35: Exemplos de filtros. a) Dois filtros atuando sobre um único grupo de elementos. b) Um único filtro atuando sobre dois grupos de elementos.

Na **Figura 36** é ilustrado o exemplo mostrado na **Figura 32** com as funções de interação geradas nesta última etapa:

Figura 36: Representação SVG associada ao documento JavaScript contendo funções que provêem interação com os elementos apresentados.

5.5 CONSIDERAÇÕES FINAIS

Neste capítulo foi descrito o processo de transformação de documentos que utilizam padrões XML visando obter, como resultado final, uma representação gráfica de

informações utilizando SVG. Foram apresentados também os componentes básicos dessa representação, e como estes podem ser estruturados e organizados. No decorrer do capítulo foi discutido também como as características de expansão progressiva e referência intrínseca são tratadas pela iVIEW.

A infra-estrutura iVIEW apresenta algumas limitações. Um exemplo é a quantidade de textos que podem ser mostrados na tela. Como SVG utiliza gráficos vetoriais para exibir objetos quaisquer, não é possível a quebra automática de linhas ao exibir textos de comprimento maior que a largura do espaço que lhes foi reservado. Isto restringe a quantidade e o conteúdo dos atributos de cada elemento que podem ser exibidos. Os elementos em si, entretanto, podem ser exibidos em grandes quantidades, pois são representados por objetos gráficos.

Além das limitações, diversas dificuldades foram encontradas durante a implementação da infra-estrutura. Para suportar documentos XML quaisquer como fonte de informação foi necessária a inclusão de etapas intermediárias até a obtenção da representação SVG final, assim como a elaboração de documentos auxiliares, como o documento XML para especificação do layout. Criar documentos XSLT genéricos que geram documentos XSLT específicos também exige uma série de cuidados, uma vez que ambos os documentos contêm a mesma sintaxe de construção e não devem, de forma alguma, se misturar durante o processo de transformação. Por fim, a criação de um documento XSLT para a geração automática de documentos JavaScript também foi necessária, pois as funções de interação dos usuários com os elementos de visualização utilizam diversos parâmetros, os quais dependem das características de cada elemento.

Apesar das dificuldades encontradas, o esforço direcionado para a criação de documentos XSLT responsáveis pela geração automática de outros documentos oferece algumas vantagens, pois tira do desenvolvedor a necessidade de ter conhecimentos sobre XSLT, SVG e JavaScript.

Os padrões XSLT e SVG são bastante extensos, principalmente quando são utilizados em conjunto com outros padrões XML. Outros recursos ainda podem ser estudados e explorados.

No próximo capítulo será apresentado um exemplo de utilização da infra-estrutura iVIEW em um ambiente que utiliza computação ubíqua para captura de informações, o **iClass**.

Capítulo 6 – Exemplo de Utilização da iVIEW no Ambiente iClass

6.1 CONSIDERAÇÕES INICIAIS

Ambientes que utilizam captura e acesso de informações contêm repositórios de dados que crescem ao longo do tempo. Em geral, o processo de captura ocorre durante as mais variadas atividades, com a participação de diversos colaboradores. Torna-se útil que a visualização do conjunto de informações geradas ofereça aos usuários destes ambientes a possibilidade de observar como as informações foram geradas ao longo do tempo, ou seja, como se dá o encadeamento sequencial das informações. Visto ainda que, nesse conjunto, possam haver várias informações que apresentam características comuns, mesmo pertencendo a grupos distintos, a possibilidade de se observar essas referências intrínsecas pode enriquecer a visualização.

Neste capítulo será apresentado um exemplo de como a utilização da iVIEW pode prover a visualização de grandes quantidades de dados sequencialmente encadeados e relacionados entre si, em um ambiente para captura de experiências em salas de aula em constante atividade.

6.2 REPRESENTAÇÃO SVG DE INFORMAÇÕES EXTRAÍDAS DO AMBIENTE ICLASS

O ambiente iClass é uma nova versão do eClass, apresentado no Capítulo 3. Esta versão foi implementada no contexto do Projeto InCA-SERVE e vem sendo utilizada no ICMC desde o 2º semestre de 2002.

No ambiente iClass, as informações referentes a cada sessão capturada ficam armazenadas em documentos XML, que são mantidos em um banco de dados da infra-estrutura que dá suporte a este ambiente. Estes documentos XML, por sua vez, contêm elementos cujos valores são dados relativos a cada sessão: pessoa responsável, local, data, curso a que

pertence e assunto tratado. A **Figura 37** é um exemplo de instância do documento XML de sessões capturadas no iClass:

```
<?xml version="1.0" encoding="UTF-8"?>
<session id="143" type="2" date="2002.10.03">
  <context_info>
    <person>Teste</person>
    <place>Lab-4</place>
    <eclass_context>
      <term>2o. 2002</term>
      <course_name>Hipermidia</course_name>
      <class_title>www2002_web</class_title>
    </eclass_context>
  </context_info>
  <chatml/>
  <whiteboardml board_width="640" board="480"/>
</session>
```

Figura 37: Instância do documento XML de sessões capturadas (session.xml).

Para dar início à primeira etapa do processo de obtenção de visualização de informação em SVG na iVIEW, as informações sobre sessões capturadas contidas nos documentos XML de sessão do iClass foram extraídas do banco de dados que as armazena e reunidas em um único documento XML através de algoritmos escritos em PHP. O novo documento – *aulasiclass.xml* – serviu como fonte para a geração da sua representação SVG final e é organizado conforme pode ser observado nos trechos desse documento mostrados a seguir. Em termos de sua estrutura, a raiz do documento é o elemento *aulas*. O elemento que contém atributos com informações sobre sessões capturadas é o elemento *aula*, que por sua vez está agrupado de acordo com o ano e o semestre correspondentes.

```
<?xml version="1.0" encoding="UTF-8" ?>
<aulas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="layout.xsd">
  <sem2_2002>
    <aula pessoa="Clarice" local="Lab-4" cursoid="Hipermidia"
    assunto="IDMEF" data="13/09/2002" tipo="expositiva"/>
    <aula pessoa="Ana" local="Lab-4" cursoid="TeciadaComputacaoLEF">
```

```

        assunto="ExercLingFormais" data="22/09/2002" tipo="expositiva"/>
        <aula pessoa="Clarice" local="Lab-4" cursoid="Hipermidia"
        assunto="KM2-mídias" data="23/09/2002" tipo="expositiva"/>
        :
    </sem2_2002>
    <sem1_2003>
        <aula pessoa="Gilberto" local="Lab-4" cursoid="HCI" assunto="CSS"
        data="21/01/2003" tipo="expositiva"/>
        <aula pessoa="Gilberto" local="Lab-4" cursoid="HCI"
        assunto="topTenMistakes" data="11/02/2003" tipo="expositiva"/>
        :
        <aula pessoa="Rosa" local="Lab-4" cursoid="Multimidia"
        assunto="topTenMistakes" data="21/03/2003" tipo="expositiva"/>
    </sem1_2003>
    <sem2_2003>
        :
    </sem2_2003>
</aulas>

```

Dado que o iVIEW é um ambiente que está em uso há pouco tempo, tendo capturado até o momento cerca de 90 sessões, foram acrescentados ao documento `aulasiclass.xml` dados que simulam aulas que ainda irão ocorrer, baseando-se nas características dos dados já existentes. Após a criação do documento `aulasiclass.xml` final, o documento `layout.xml` correspondente foi elaborado e está estruturado como mostra o trecho a seguir:

```

<?xml version="1.0" encoding="UTF-8"?>
<especificacao>
<layout>
    <janela largura="770" altura="480" cor.de.fundo="#E2ECF5"/>
    <titulo label="Sessões Capturadas - 2o. sem 2002 / 2o.sem 2003"
    fonte face="verdana" fonte tamanho="16" fonte.cor="url(#Gradient3)"/>
    :
    <barras>
        <barra nome="session" espessura="8" altura="20"/>
    </barras>
</layout>
<elementos>
    <elemento caminho="aulas/sem2_2002" grupoid="sess1">
        :
    </elemento>

```

```

        <representacao nome="session" stroke_cor="url(#Gradient1)"
        cor="url(#Gradient1)"/>
        <posicao_inicial min="1" max="36" x="19" y="60"/>
        <posicao_inicial min="37" max="72" x="19" y="85"/>
        <deslocamento>15</deslocamento>
        <atributos>
            <atributo nome="pessoa"/>
            <atributo nome="local"/>
            <atributo nome="cursoid"/>
            <atributo nome="assunto"/>
            <atributo nome="data"/>
        </atributos>
    </elemento>

    :

</elementos>
<filtros presente="sim">

    :

    <filtro nome="cursos" descricao="Disciplinas/Atividades" fonte_cor="black"
    fonte_tamanho="12" x="550" y="0">
        <grupoelem grupoid="sess1" atributoid="2"/>
        <grupoelem grupoid="sess2" atributoid="2"/>
        <grupoelem grupoid="sess3" atributoid="2"/>
        <item label="Hipermidia" cor="#333333" tamanho="11"/>
        <item label="TeoriadaComputacaoLF" cor="#333333" tamanho="11"/>
        <item label="SeminariosdeHipermidia" cor="#333333" tamanho="11"/>
        :
    </filtro>

    :

</filtros>
</especificacao>

```

Nesse trecho do documento `layout.xml`, podemos observar algumas especificações de apresentação definidas:

- ❑ Janela com 770 pixels de largura por 480 pixels de altura;
- ❑ As sessões capturadas são representadas por barras de nome “session”, com 8 pixels de largura e 20 pixels de altura;
- ❑ O primeiro grupo de elementos de visualização é apresentado em duas fileiras, cujas posições são definidas pelo elemento *posicao_inicial*;

- ❑ O deslocamento entre os elementos é de 15 pixels;
- ❑ São mostrados cinco atributos dos elementos, que neste exemplo correspondem a sessões capturadas no iClass: *pessoa*, *local*, *curso*, *assunto* e *data*.
- ❑ O primeiro filtro seleciona as sessões capturadas de acordo com a disciplina ou atividade a que pertencem.

Como resultado, foi gerada a representação SVG contida na **Figura 38**. A área de exibição de elementos de visualização contém três grupos de sessões capturadas, divididos em semestre e ano. A ordem de apresentação dos elementos que compõem cada grupo foi estabelecida de acordo com a data de ocorrência das sessões. Os dados sobre as sessões do 2º. semestre de 2002 foram obtidos a partir do banco de dados do ambiente iClass. Os dados sobre as sessões do 1º. e 2º. semestres de 2003 foram acrescentados ao documento XML fonte para simular a visualização das sessões capturadas no iClass ao longo de um período mais extenso.

Figura 38: Representação SVG de sessões capturadas no iClass, do 2º. semestre de 2002 ao 2º. semestre de 2003

Para esta aplicação foram utilizados filtros de seleção para discriminar as sessões capturadas quanto à disciplina ou à atividade a que pertencem, pessoa responsável, e local de ocorrência da captura. Todos os filtros podem atuar simultaneamente sobre os grupos de elementos. A **Figura 39** apresenta o resultado da interação de um usuário da aplicação com a representação SVG das sessões capturadas no iClass. O atributo *assunto* foi escolhido neste exemplo para estabelecer o relacionamento entre as sessões. Ainda na **Figura 39**, os filtros de seleção foram utilizados para que fossem exibidos apenas sessões correspondentes às disciplinas “Hipermedia”, “Seminarios em Hipermedia” e “SCE5844 Seminarios Avancados em Hipermedia”, com “Maria” como pessoa responsável e “Lab-4” como local de captura. São destacadas na visualização, como resposta ao clique do mouse, todas as sessões cujo assunto é “SeminarioAdriane”.

Figura 39: Representação SVG de sessões capturadas após a interação de um usuário com funções contidas em um documento JavaScript.

6.3 CONSIDERAÇÕES FINAIS

Com este exemplo, observou-se na prática que a utilização de documentos estruturados torna possível a geração automática de uma representação gráfica em SVG correspondente a informações sobre sessões capturadas. Uma das principais vantagens que o uso da infraestrutura iVIEW oferece para a geração dessa representação é que a mesma especificação de layout pode ser utilizada, posteriormente, por diversas instâncias XML com dados sobre sessões capturadas.

No próximo capítulo as conclusões finais desta dissertação são apresentadas.

Capítulo 7 – Conclusões

A rede mundial de computadores – a Internet – é tida hoje como um dos maiores repositórios de informações dentro dos mais diversos domínios de aplicações. No entanto, para os usuários finais, a tarefa de localizar, interpretar e interagir com um vasto universo de informações pode tornar-se pouco intuitiva. A Visualização de Informação é uma área que surgiu com o objetivo de prover formas interativas de representação de dados, através de mecanismos suportados por computador.

Através das pesquisas realizadas durante este trabalho, constatou-se que a grande maioria das aplicações voltadas à Visualização de Informação direciona seus esforços para organizar e apresentar de dados de forma rápida e interativa. No entanto, existe o fato de que os dados podem evoluir e ter muitas informações embutidas, as quais nem sempre são visíveis através de métodos comuns de apresentação de dados.

Além disso, diversas técnicas e ferramentas de visualização estão direcionadas para aplicações que utilizam tipos de dados específicos e que, portanto, oferecem suporte restrito à visualização de dados com características evolucionárias.

Considerando-se características como a expansão progressiva e o relacionamento intrínseco inerentes à informação evolucionária, foi definida e implementada uma infra-estrutura para suporte à visualização desse tipo de informação, visando especialmente os usuários da Web: a infra-estrutura iVIEW. A infra-estrutura explora os padrões XML, XML Schema, XSLT e SVG em termos de suas características enquanto linguagens estruturadas e integradas para as quais existe um vasto ferramental disponível. A infra-estrutura iVIEW permite a desenvolvedores definir representações gráficas SVG para itens de informações de modo que seus usuários possam ver de forma interativa grandes quantidades de elementos de informação em uma única visualização. Com os processos de geração automática de instruções para transformação e representação de dados que a infra-estrutura oferece, desenvolvedores não precisam ter conhecimento dos diversos padrões de linguagem utilizadas, com exceção das linguagens XML e XML Schema.

A infra-estrutura iVIEW está sendo complementada com uma documentação correspondente a um manual do usuário. Além disso, está em andamento a integração da infra-estrutura com serviços do Projeto InCA-SERVE.

Dado que os padrões XML são consideravelmente extensos, houve a preocupação, neste trabalho, de se produzir uma primeira versão da infra-estrutura iVIEW. Algumas possibilidades para se estender os recursos oferecidos pela infra-estrutura atual são:

- investigar meios para que o relacionamento entre informações possa ser estabelecido após a geração da representação SVG;
- investigar outros recursos que SVG é capaz de prover e, assim, acrescentar novas funções de interação;
- criar editores gráficos para definir o layout de visualização.

Com este trabalho, o objetivo de prover a visualização de informações considerando seu crescimento ao longo do tempo e o relacionamento existente entre elas foi atingido, em especial dentro do contexto da World Wide Web.

Referências Bibliográficas

[Abowd, 1999] ABOWD, G. Classroom 2000: An experiment with the instrumentation of a living educational environment. IBM Systems Journal, v. 38, n.4, p. 508-530, out. 1999.

[Abowd et al., 1998a] ABOWD, G.; ATEKESON, C.; BROTHERTON, J; ENQVIST, T.; GULLY, P.; LEMON, J. Investigating the capture, integration and access problem of ubiquitous computing in educational setting. Proc. of the CHI'98, ACM, New York, p. 440-447, 1998.

[Abowd et al., 1998b] ABOWD, G.; BROTHERTON, J; BHALODIA, J. Classroom 2000: A System for Capturing and Accessing Multimedia Classroom Experiences. Proceedings of CHI '98, p. 20-21, mai. 1998.

[Abowd et al., 2000] ABOWD, D.; LONNIE, D.; BROTHERTON, J. Building a Digital Library of Captured Educational Experiences. International Conference on Digital Libraries, Kyoto, Japan, p. 13-16, nov. 2000.

[Brotherton et al., 1998] BROTHERTON, J; BHALODIA, J.; ABOWD, G. Automated Capture, Integration, and Visualization of Multiple Media Streams. Proceedings of IEEE Multimedia'98, p.54-63, jul. 1998.

[Brotherton et al., 1999] BROTHERTON, J; ABOWD, G.; TRUONG, K. Supporting Capture and Access Interfaces for Informal and Opportunistic Meetings. Georgia Institute of Technology Technical Report GIT-GVU-99-06, 1999.

[Brotherton & Abowd, 1998] BROTHERTON, J; ABOWD, G. Rooms Take Note: Room Take Notes! In Working Papers of AAAI'98 Spring Symposium, mar. 1998.

[Card et al., 1996] CARD, S. K.; ROBERTSON, G. G.; YORK, W. The WebBook and the Web Forager: An Information Workspace for the World-Wide Web. Proceedings of the ACM Human Factors in Computing Systems, p. 111-ff, 1996.

[Card et al., 1999] CARD, S. K.; MACKINLAY J. D.; SHNEIDERMAN, B. Readings in Information Visualization Using Vision to Think. Morgan Kaufmann Publishers, Inc, p. 686, 1999.

[Cattelan et al., 2002] CATTELAN, R. G.; ANDRADE, A. R.; PENEDO, C. F. R.; PIMENTEL, M. G. P. IClass: um sistema para captura e acesso de sessões em ambiente educacional, REIC - Revista eletrônica de Iniciação Científica, 2002.

[Chi et al., 1998] CHI, E. H.; MACKINLAY, J.; PIROLI, P.; GOSSWEILER, R.; CARD, S. K. Visualizing the Evolution of Web Ecologies. Proceedings of ACM SIGCHI '98, Los Angeles, CA, p. 400-407, abr. 1998.

[Davis et al., 1999] DAVIS, R. C.; LANDAY, J. A.; CHEN, V.; HUANG, J.; LEE, R. B.; LI, F.; LIN, J.; MORREY III, C. B.; SCHLEIMER, B.; Price, M. N.; SCHLIT, B. N. NotePals: Lightweight Note Sharing by the Group, for the Group. Proceedings of Human Factors in Computer Systems: CHI 99, Pittsburgh, PA, p. 338-345, mai. 1999.

[Dey et al., 1999] DEY, A. K.; SALBER, D.; ABOWD, G. D.; FUTAKAWA, M. The Conference Assistant: Combining Context-Awareness with Wearable Computing. Proceedings of ISWC, p. 21-28, 1999.

[Eick et al., 1992] EICK S.G.; STEFFEN, J.L.; SUMNER, E.E. SeeSoft: A tool for visualizing line oriented software. IEEE Transactions on Software Engineering, p. 11-18, 1992.

[Freeman & Gelernter, 1996] FREEMAN, E.; GELERNTER., D., LifeStreams: A storage model for personal data, ACM SIGMOD Bulletin 25, 1, p. 80-86, mar. 1996.

[Hearst et al., 1996] HEARST, M.; KOPEC, G.; BROTSKY, D. Research in support of Digital libraries at Xerox/PARC. Part II: paper and digital documents. D-Lib Magazine, ISSN 1082-9873, jun. 1996.

[Hibino & Rundensteiner, 1996] HIBINO, S.; RUNDENSTEINER, E. A. MMVIS: A MultiMedia Visual Information Seeking Environment for Video Analysis. ACM Multimedia'96 Conference Proceedings, p. 15-16, nov. 1996.

[Hong & Landay, 2001] HONG, J. I.; LANDAY, J. A. WebQuilt: A Framework for Capturing and Visualizing the Web Experience. Proceedings of The Tenth International World Wide Web Conference (WWW10), Hong Kong, p. 717-724, abr. 2001

[Jerding & Stasko, 1995] JERDING, D. F.; STASKO, J. T. The Information Mural: A technique for displaying and navigating large information space. Proceedings of the IEEE Visualization , Atlanta, GA, p. 43-50, out. 1995.

[Johnson & Shneiderman, 1991] JOHNSON, B.; SHNEIDERMAN, B. Tree-maps: A space filling approach to the visualization of hierarchical information structures. Proceedings of IEEE Visualization '91, p. 284-291, out. 1991.

[Mackinlay et al, 1991] MACKINLAY, J.; ROBERTSON, G.; CARD, S. The perspective wall: detail and context smoothly integrated. In Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems, New Orleans, Louisiana, p.173-179, 1991.

[Pimentel & Abowd, 1999] **PIMENTEL**, M. G. C. & ABOWD, G. Development and understanding of automated capture environments to support long-term use. Projeto de Cooperação Internacional Aprovado junto ao ProTeM-Cc-Cnpq/Brasil e ao NSF/EUA, 99.

[Pimentel, 2000] **PIMENTEL**, M. G. C. Modelagem, Projeto e Implementação da Infra-estrutura SERVE. Auxílio Pesquisa Fapesp 00/12658-0, 2000.

[Pimentel et al., 2000a] **PIMENTEL**, M. G. C.; ABOWD, G.; ISHIGURO, I. Linking by Interacting: a Paradigm for Authoring Hypertext. Proc. ACM Hypertext'00, San Antonio, TX, p. 39-48, mai. 2000.

[Pimentel et al., 2000b] **PIMENTEL**, M. G. C.; ABOWD, G.; KERIMBAEV, B.; ISHIGURO, Y.; GUZDIAL, M. Supporting Long-Term Educational Activities Through Dynamic Web Interfaces. Journal Interacting With Computers, p. 353-374, 2000.

[Pimentel & Teixeira, 2000] **PIMENTEL**, M. G. C. & TEIXEIRA, C. A. C. XML: Explorando suas aplicações na Web. JAI2000-SBC, XIX Jornada de Atualização em Informática.

- [Rao & Card, 1994] RAO, R.; CARD, S. K. The Table Lens: Merging Graphical and Symbolic Representations In an Interactive Focus + Context Visualization for Tabular Information. Proceedings of ACM Human Factors in Computing Systems, p. 318-322, 1994.
- [Richter et al., 2001] RICHTER, H.; ABOWD, G. D.; GEYER, W.; FUCHS, L.; DALIAVAD, S.; POLTROCK, S. Integrating Meeting Capture within a Collaborative Team Environment. Ubicomp 2001: Ubiquitous Computing, p. 123-138, set. 2001.
- [Robertson et al., 1991] ROBERTSON, G. G.; MACKINLAY, J. D.; CARD, S. Cone Trees: Animated 3D Visualizations of Hierarchical Information. Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems, p. 189-194, abr. 1991.
- [Shneiderman, 1996] SHNEIDERMAN, B. The eyes have it: A task by data type taxonomy of information visualizations. Proc. IEEE Symposium on Visual Languages'96, IEEE, Los Alamos, CA, p. 336-343, set. 1996.
- [Shneiderman et al., 1996a] NORTH, C.; PLAISANT, C.; SHNEIDERMAN, B. User Controlled Overviews of an Image Library: A Case Study of the Visible Human. Proc. ACM Digital Libraries '96 Conf, ACM Press, p. 74-82, 1996.
- [Shneiderman et al., 1996b] PLAISANT, C.; MILASH, B.; ROSE, A.; WIDOFF, S.; SHNEIDERMAN, B. LifeLine: Visualizing personal histories. ACM CHI'96 Conference, p. 221-227, abr. 1996.

Apêndice A

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="especificacao">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="layout" type="layoutType" maxOccurs="1"/>
        <xsd:element name="infobox" type="infoboxType" maxOccurs="1"/>
        <xsd:element name="elementos" type="elementosType" minOccurs="1"
maxOccurs="1"/>
        <xsd:element name="filtros" type="filtrosType" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="layoutType">
    <xsd:sequence>
      <xsd:element name="janela" type="janelaType" maxOccurs="1"/>
      <xsd:element name="titulo" type="tituloType" maxOccurs="1"/>
      <xsd:element name="script" type="scriptType" maxOccurs="1"/>
      <xsd:element name="textos" type="textosType" minOccurs="0"
maxOccurs="100"/>
      <xsd:element name="cores_gradiente" type="cores_gradienteType"
minOccurs="0" maxOccurs="100"/>
      <xsd:element name="barras" type="barrasType" minOccurs="0"
maxOccurs="20"/>
      <xsd:element name="circulos" type="circulosType" minOccurs="0"
maxOccurs="20"/>
      <xsd:element name="retangulos" type="retangulosType" minOccurs="0"
maxOccurs="20"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="janelaType">
    <xsd:attribute name="largura" type="xsd:string" use="required"/>
    <xsd:attribute name="altura" type="xsd:string" use="required"/>
    <xsd:attribute name="cor_de_fundo" type="xsd:string"
use="required"/>
  </xsd:complexType>

  <xsd:complexType name="tituloType">
    <xsd:attribute name="label" type="xsd:string" use="required"/>
    <xsd:attribute name="fonte_face" type="xsd:string" use="required"/>
    <xsd:attribute name="fonte_tamanho" type="xsd:string"
use="required"/>
    <xsd:attribute name="fonte_cor" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="scriptType">
    <xsd:attribute name="arquivo" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="textosType">
    <xsd:sequence>
      <xsd:element name="texto" type="textoType" maxOccurs="20"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="textoType">
    <xsd:attribute name="label" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:schema>
```

```

        <xsd:attribute name="x" type="xsd:string" use="required"/>
        <xsd:attribute name="y" type="xsd:string" use="required"/>
        <xsd:attribute name="fonte_face" type="xsd:string" use="required"/>
        <xsd:attribute name="fonte_tamanho" type="xsd:string"
use="required"/>
        <xsd:attribute name="fonte_cor" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="cores_gradienteType">
    <xsd:sequence>
        <xsd:element name="gradiente" type="gradienteType" maxOccurs="20"/>
    </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="gradienteType">
        <xsd:attribute name="nome" type="xsd:string" use="required"/>
        <xsd:attribute name="x1" type="xsd:string" use="required"/>
        <xsd:attribute name="x2" type="xsd:string" use="required"/>
        <xsd:attribute name="y1" type="xsd:string" use="required"/>
        <xsd:attribute name="y2" type="xsd:string" use="required"/>
    <xsd:sequence>
        <xsd:element name="stop" type="stopType" minOccurs="2"
maxOccurs="20"/>
    </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="stopType">
        <xsd:attribute name="offset" type="xsd:string" use="required"/>
        <xsd:attribute name="stop_color" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="barrasType">
    <xsd:sequence>
        <xsd:element name="barra" type="barraType" maxOccurs="20"/>
    </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="barraType">
        <xsd:attribute name="nome" type="xsd:string" use="required"/>
        <xsd:attribute name="espessura" type="xsd:string" use="required"/>
        <xsd:attribute name="altura" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="circulosType">
    <xsd:sequence>
        <xsd:element name="circulo" type="circuloType" maxOccurs="20"/>
    </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="circuloType">
        <xsd:attribute name="nome" type="xsd:string" use="required"/>
        <xsd:attribute name="raio" type="xsd:string" use="required"/>
        <xsd:attribute name="stroke_espessura" type="xsd:string"
use="required"/>
    </xsd:complexType>

    <xsd:complexType name="retangulosType">
    <xsd:sequence>
        <xsd:element name="retangulo" type="retanguloType" maxOccurs="20"/>
    </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="retanguloType">

```



```

        <xsd:attribute name="nome" type="xsd:string" use="required"/>
        <xsd:attribute name="largura" type="xsd:string" use="required"/>
        <xsd:attribute name="altura" type="xsd:string" use="required"/>
        <xsd:attribute name="stroke_espessura" type="xsd:string"
use="required"/>
    </xsd:complexType>

    <xsd:complexType name="infoboxType">
    <xsd:sequence>
        <xsd:element name="max_caracteres" type="xsd:string"
maxOccurs="1"/>
        <xsd:element name="fonte" type="fonteType" maxOccurs="1"/>
    </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="fonteType">
        <xsd:attribute name="cor" type="xsd:string" use="required"/>
        <xsd:attribute name="face" type="xsd:string" use="required"/>
        <xsd:attribute name="tamanho" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="elementosType">
    <xsd:sequence>
        <xsd:element name="elemento" type="elementoType" maxOccurs="20"/>
    </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="elementoType">
        <xsd:attribute name="caminho" type="xsd:string" use="required"/>
        <xsd:attribute name="gruroid" type="xsd:string" use="required"/>
    <xsd:sequence>
        <xsd:element name="area" type="areaType" minOccurs="1"/>
        <xsd:element name="titulo" type="etituloType" minOccurs="1"/>
        <xsd:element name="comandos" type="comandosType" minOccurs="1"/>
        <xsd:element name="representacao" type="representacaoType"
minOccurs="1"/>
        <xsd:element name="posicao_inicial" type="posicao_inicialType"
minOccurs="1" maxOccurs="50"/>
        <xsd:element name="deslocamento" type="xsd:string" minOccurs="1"/>

        <xsd:element name="atributos" type="atributosType" minOccurs="1"/>
    </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="areaType">
        <xsd:attribute name="x" type="xsd:string" use="required"/>
        <xsd:attribute name="y" type="xsd:string" use="required"/>
        <xsd:attribute name="largura" type="xsd:string" use="required"/>
        <xsd:attribute name="altura" type="xsd:string" use="required"/>
        <xsd:attribute name="stroke_espessura" type="xsd:string"
use="required"/>
        <xsd:attribute name="stroke_cor" type="xsd:string" use="required"/>
        <xsd:attribute name="cor_de_fundo" type="xsd:string"
use="required"/>
    </xsd:complexType>

    <xsd:complexType name="etituloType">
        <xsd:attribute name="nome" type="xsd:string" use="required"/>
        <xsd:attribute name="x" type="xsd:string" use="required"/>
        <xsd:attribute name="y" type="xsd:string" use="required"/>
        <xsd:attribute name="fonte_tamanho" type="xsd:string"
use="required"/>
        <xsd:attribute name="fonte_cor" type="xsd:string" use="required"/>
    </xsd:complexType>

```

```

<xsd:complexType name="comandosType">
<xsd:sequence>
    <xsd:element name="mouseover" type="mouseType" minOccurs="1"/>
    <xsd:element name="mouseout" type="mouseType" minOccurs="1"/>
    <xsd:element name="mousedown" type="mousedownType" minOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="mouseType">
    <xsd:attribute name="funcao" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="mousedownType">
    <xsd:attribute name="ativo" type="xsd:string" use="required"/>
<xsd:sequence>
    <xsd:element name="parametros" type="parametrosType"/>
    <xsd:element name="grupoelem" type="grupoelemType" maxOccurs="20"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="parametrosType">
    <xsd:attribute name="atr_entrada" type="xsd:string"
use="required"/>
    <xsd:attribute name="stroke_cor" type="xsd:string" use="required"/>
    <xsd:attribute name="cor" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="grupoelemType">
    <xsd:attribute name="grupoid" type="xsd:string" use="required"/>
    <xsd:attribute name="atributoid" type="xsd:string" use="required"/>
    <xsd:attribute name="stroke_cor original" type="xsd:string"
use="required"/>
    <xsd:attribute name="cor_original" type="xsd:string"
use="required"/>
    <xsd:attribute name="stroke_cor" type="xsd:string" use="required"/>
    <xsd:attribute name="cor" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="representacaoType">
    <xsd:attribute name="nome" type="xsd:string" use="required"/>
    <xsd:attribute name="stroke_cor" type="xsd:string" use="required"/>
    <xsd:attribute name="cor" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="posicao_inicialType">
    <xsd:attribute name="min" type="xsd:string" use="required"/>
    <xsd:attribute name="max" type="xsd:string" use="required"/>
    <xsd:attribute name="x" type="xsd:string" use="required"/>
    <xsd:attribute name="y" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="atributosType">
<xsd:sequence>
    <xsd:element name="atributo" type="atributoType" minOccurs="1"
maxOccurs="5"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="atributoType">
    <xsd:attribute name="nome" type="xsd:string" use="required"/>
</xsd:complexType>

```

```

<xsd:complexType name="filtroType">
  <xsd:attribute name="presente" type="xsd:string" use="required"/>
<xsd:sequence>
  <xsd:element name="titulo" type="titulo3Type" minOccurs="1"/>
  <xsd:element name="area" type="areaType" minOccurs="1"/>
  <xsd:element name="filtro" type="filtroType" maxOccurs="20"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="titulo3Type">
  <xsd:attribute name="label" type="xsd:string" use="required"/>
  <xsd:attribute name="fonte_tamanho" type="xsd:string"
use="required"/>
  <xsd:attribute name="fonte_cor" type="xsd:string" use="required"/>
  <xsd:attribute name="x" type="xsd:string" use="required"/>
  <xsd:attribute name="y" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="filtroType">
  <xsd:attribute name="nome" type="xsd:string" use="required"/>
  <xsd:attribute name="descricao" type="xsd:string" use="required"/>
  <xsd:attribute name="fonte_cor" type="xsd:string" use="required"/>
  <xsd:attribute name="fonte_tamanho" type="xsd:string"
use="required"/>
  <xsd:attribute name="x" type="xsd:string" use="required"/>
  <xsd:attribute name="y" type="xsd:string" use="required"/>
<xsd:sequence>
  <xsd:element name="grupoitem" type="grupoitem2Type" minOccurs="1"
maxOccurs="20"/>
  <xsd:element name="item" type="itemType" minOccurs="1"
maxOccurs="20"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="grupoitem2Type">
  <xsd:attribute name="grupooid" type="xsd:string" use="required"/>
  <xsd:attribute name="atributooid" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="itemType">
  <xsd:attribute name="label" type="xsd:string" use="required"/>
  <xsd:attribute name="cor" type="xsd:string" use="required"/>
  <xsd:attribute name="tamanho" type="xsd:string" use="required"/>
</xsd:complexType>

</xsd:schema>

```

Apêndice B

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:outputxsl="replacement">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
omit-xml-declaration="no" standalone="no"></xsl:output>

<xsl:namespace-alias stylesheet-prefix="outputxsl" result-prefix="xsl"/>

<xsl:strip-space elements="*" /><!-- without this line-breaks appear as
text nodes -->

<xsl:template match="/">
<outputxsl:stylesheet version="1.1"
xmlns:xlink="http://www.w3.org/1999/xlink">
<outputxsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes" omit-xml-declaration="no" standalone="no" doctype-public="
//W3C//DTD SVG 20001102//EN" doctype
system="http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd"
media-type="text/svg">
</outputxsl:output>

<outputxsl:strip-space elements="*" />

<outputxsl:template match="/">
<xsl:variable name="width"
select="especificacao/layout/janela/@largura"/>
<xsl:variable name="height"
select="especificacao/layout/janela/@altura"/>
<svg width="{width}" height="{height}" onload="onload(evt)">
<xsl:variable name="bgcolor"
select="especificacao/layout/janela/@cor_de_fundo"/>
<rect x="0" y="0" width="100%" height="100%" fill="{bgcolor}"/>
<xsl:variable name="script"
select="especificacao/layout/script/@arquivo"/>
<script type="text/javascript" xlink:href="{script}"/>
<xsl:variable name="tfiface"
select="especificacao/layout/titulo/@fonte_face"/>
<xsl:variable name="tfsize"
select="especificacao/layout/titulo/@fonte_tamanho"/>
<xsl:variable name="tfcolor"
select="especificacao/layout/titulo/@fonte_cor"/>
<text font-family="{tfiface}" font-size="{tfsize}" fill="{tfcolor}"
x="10" y="20"><xsl:value-of
select="especificacao/layout/titulo/@label"/></text>

<defs>
<xsl:for-each select="especificacao/layout/cores_gradiente/gradiente">
<xsl:variable name="idgrad" select="@nome"/>
<xsl:variable name="x1" select="@x1"/>
<xsl:variable name="x2" select="@x2"/>
<xsl:variable name="y1" select="@y1"/>
<xsl:variable name="y2" select="@y2"/>
<linearGradient id="{idgrad}" gradientUnits="userSpaceOnUse" x1="{x1}"
x2="{x2}" y1="{y1}" y2="{y2}">
<xsl:for-each select="stop">
<xsl:variable name="offset" select="@offset"/>
<xsl:variable name="stopcolor" select="@stop_color"/>
<stop offset="{offset}" stop-color="{stopcolor}"/>
</xsl:for-each>

```

```

</linearGradient>
</xsl:for-each>
<xsl:for-each select="especificacao/layout/barras/barra">
  <xsl:variable name="nome" select="@nome"/>
  <xsl:variable name="altura" select="@altura"/>
  <xsl:variable name="espessura" select="@espessura"/>
  <symbol id="{ $nome}" overflow='visible'>
    <line y2="{ $altura}" stroke-width="{ $espessura}" />
  </symbol>
</xsl:for-each>
<xsl:for-each select="especificacao/layout/circulos/circulo">
  <xsl:variable name="nome" select="@nome"/>
  <xsl:variable name="raio" select="@raio"/>
  <xsl:variable name="stroke" select="@stroke_espessura"/>
  <symbol id="{ $nome}" overflow='visible'>
    <circle cy="0" r="{ $raio}" stroke-width="{ $stroke}" />
  </symbol>
</xsl:for-each>
<xsl:for-each select="especificacao/layout/retangulos/retangulo">
  <xsl:variable name="nome" select="@nome"/>
  <xsl:variable name="stroke" select="@stroke_espessura"/>
  <xsl:variable name="altura" select="@altura"/>
  <xsl:variable name="largura" select="@largura"/>
  <symbol id="{ $nome}" overflow='visible'>
    <rect width="{ $largura}" height="{ $altura}" stroke-width="{ $espessura}" />
  </symbol>
</xsl:for-each>
</defs>

<xsl:for-each select="especificacao/layout/textos/texto">
  <xsl:variable name="txtx" select="@x"/>
  <xsl:variable name="txty" select="@y"/>
  <xsl:variable name="txtface" select="@fonte_face"/>
  <xsl:variable name="txtcolor" select="@fonte_cor"/>
  <xsl:variable name="txtsize" select="@fonte_tamanho"/>
  <text font-family="{ $txtface}" font-size="{ $txtsize}" fill="{ $txtcolor}"
  x="{ $txtx}" y="{ $txty}"><xsl:value-of select="@label"/></text>
</xsl:for-each>

<xsl:if test="especificacao/filtros/@presente = 'sim'">
  <xsl:variable name="tx" select="especificacao/filtros/titulo/@x"/>
  <xsl:variable name="ty" select="especificacao/filtros/titulo/@y"/>
  <xsl:variable name="fsize"
  select="especificacao/filtros/titulo/@fonte_tamanho"/>
  <xsl:variable name="fcolor"
  select="especificacao/filtros/titulo/@fonte_cor"/>
  <xsl:variable name="ax" select="especificacao/filtros/area/@x"/>
  <xsl:variable name="ay" select="especificacao/filtros/area/@y"/>
  <xsl:variable name="awidth"
  select="especificacao/filtros/area/@largura"/>
  <xsl:variable name="aheight"
  select="especificacao/filtros/area/@altura"/>
  <xsl:variable name="astroke"
  select="especificacao/filtros/area/@stroke_espessura"/>
  <xsl:variable name="astroke_c"
  select="especificacao/filtros/area/@stroke_cor"/>
  <xsl:variable name="abgcolor"
  select="especificacao/filtros/area/@cor_de_fundo"/>
  <rect x="{ $ax}" y="{ $ay}" height="{ $aheight}" width="{ $awidth}"
  fill="{ $abgcolor}" stroke="{ $astroke_c}" stroke-width="{ $astroke}" />
  <text style="font-size: { $fsize}; fill: { $fcolor}" x="{ $tx}"
  y="{ $ty}"><xsl:value-of
  select="especificacao/filtros/titulo/@label"/></text>
</xsl:if>

```

```

<outputxsl:apply-templates/>
<xsl:if test="especificacao/filtros/@presente = 'sim'">
<xsl:for-each select="especificacao/filtros/filtro">
<xsl:variable name="templatename" select="@nome"/>
<outputxsl:call-template name="{ $templatename }"/>
</xsl:for-each>
</xsl:if>
<outputxsl:call-template name="info1"/>
<outputxsl:call-template name="info2"/>
<outputxsl:call-template name="info3"/>
<outputxsl:call-template name="info4"/>
<outputxsl:call-template name="info5"/>
<outputxsl:call-template name="info6"/>
</svg>
</outputxsl:template>

<outputxsl:template name="truncate">
<xsl:variable name="max_char"
select="especificacao/infobox/max_caracteres"/>
<outputxsl:param name="theString" select="."/>
<outputxsl:variable name="stringLength" select="string-
length($theString)"/>
<outputxsl:value-of select="substring($theString,0,{ $max_char } )"/>
<outputxsl:if test="$stringLength > { $max_char }">...</outputxsl:if>
</outputxsl:template>

<xsl:for-each select="especificacao/elementos/elemento">
<xsl:variable name="path" select="@caminho"/>
<outputxsl:template match="{ $path }">
<xsl:variable name="rx" select="area/@x"/>
<xsl:variable name="ry" select="area/@y"/>
<xsl:variable name="width" select="area/@largura"/>
<xsl:variable name="height" select="area/@altura"/>
<xsl:variable name="stroke_w" select="area/@stroke_espessura"/>
<xsl:variable name="stroke_c" select="area/@stroke_cor"/>
<xsl:variable name="bgcolor" select="area/@cor_de_fundo"/>
<xsl:variable name="tx" select="titulo/@x"/>
<xsl:variable name="ty" select="titulo/@y"/>
<xsl:variable name="fontcolor" select="titulo/@fonte_cor"/>
<xsl:variable name="fontsize" select="titulo/@fonte_tamanho"/>
<rect x="{ $rx }" y="{ $ry }" height="{ $height }" width="{ $width }"
stroke="{ $stroke_c }" stroke-width="{ $stroke_w }" fill="{ $bgcolor }"/>
<text style="font-size:{ $fontsize }; fill:{ $fontcolor }" x="{ $tx }"
y="{ $ty }"><xsl:value-of select="titulo/@nome"/></text>
<xsl:variable name="mover" select="comandos/mouseover/@funcao"/>
<xsl:variable name="mout" select="comandos/mouseout/@funcao"/>
<g onmouseover="{ $mover }" onmouseout="{ $mout }">
<outputxsl:variable name="n" select="count(descendant::*)"/>
<xsl:variable name="id" select="@gruposid"/>
<count id="{ $id }">
<num><outputxsl:value-of select="{ $n }"/></num>
</count>
<outputxsl:for-each select="*">
<xsl:variable name="desloc" select="deslocamento"/>
<xsl:variable name="atr_cn"
select="comandos/mousedown/parametros/@atr_entrada"/>
<xsl:variable name="repres" select="representacao/@nome"/>
<xsl:variable name="fill" select="representacao/@cor"/>
<xsl:variable name="stroke" select="representacao/@stroke_cor"/>

<xsl:for-each select="posicao_inicial">
<xsl:variable name="dx" select="@x"/>
<xsl:variable name="dy" select="@y"/>
<xsl:variable name="min" select="@min"/>

```

```

<xsl:variable name="max" select="@max"/>

<outputxsl:if test="(count(preceding-sibling:*)+1) > ($min)-1">
<outputxsl:if test="count(preceding-sibling:*) < ($max)">

<outputxsl:variable name="deslocaX" select="(count(preceding-sibling:*)
- (($min) 1)) * ($desloc) + ($dx)"/>
<outputxsl:variable name="getid" select="count(preceding-sibling:*) +
1"/>
<xsl:variable name="getid" select="'{$getid}'"/>
<xsl:variable name="deslocaX" select="'{$deslocaX}'"/>
<g id="{($id)}{($getid)}" transform="translate({$deslocaX},{$dy})">
<outputxsl:variable name="param" select="@{($atr_en)}/>
<xsl:variable name="param" select="'{$param}'"/>
<use xlink:href="#{($repros)}" fill="{($fill)}" stroke="{($stroke)}"
onclick="showrelated{($id)}(evt,'{$param}')" visibility="visible">yes</use>
<inactivefilters></inactivefilters>
<xsl:for-each select="..atributos/atributo">
<xsl:variable name="atr" select="@none"/>
<atr><outputxsl:value-of select="@{($atr)}/></atr>
<atrtrunc><outputxsl:call-template name="truncate"><outputxsl:with-param
name="theString" select="@{($atr)}/></outputxsl:call-template></atrtrunc>
</xsl:for-each>
</g>

</outputxsl:if>
</outputxsl:if>

</xsl:for-each>

</outputxsl:for-each>
</g>
</outputxsl:template>
</xsl:for-each>

<xsl:if test="especificacao/filtros/@presente = 'sim'">
<xsl:for-each select="especificacao/filtros/filtro">
<xsl:variable name="templatename" select="@none"/>
<outputxsl:template name="{($templatename)}">
<xsl:variable name="fx" select="@x"/>
<xsl:variable name="fy" select="@y"/>
<g transform="translate({$fx},{$fy})">
<xsl:variable name="fs" select="@fonte_tamanho"/>
<xsl:variable name="fc" select="@fonte_cor"/>
<text style="font-size:{$fs}; fill:{$fc}" x="18" y="45"><xsl:value-of
select="@descricao"/></text>
<xsl:variable name="filterid" select="count(preceding-sibling::filtro)"/>
<xsl:for-each select="item">
<xsl:variable name="isize" select="@tamanho"/>
<xsl:variable name="icolor" select="@cor"/>
<xsl:variable name="Yposition" select="count(preceding-sibling::item) *
15 + 60"/>
<text style="font-size:{$isize}; fill:{$icolor}" x="40"
y="{($Yposition)}"><xsl:call-template name="truncate"><xsl:with-param
name="theString" select="@label"/></xsl:call-template></text>
<xsl:variable name="itemid" select="count(preceding-sibling::item)"/>
<xsl:variable name="Y_rectpos" select="count(preceding-sibling::item) *
15 + 50"/>
<xsl:variable name="label" select="@label"/>
<a><rect id="cbx{($filterid)}_{($itemid)}"
onclick="show{($templatename)}(evt,'{$label}')"
style="fill:blue;stroke:black;stroke-width:1" x="20" y="{($Y_rectpos)}"
width="10" height="10">yes</rect></a>
</xsl:for-each>
</g>

```



```

</outputxsl:template>
</xsl:for-each>
</xsl:if>

<xsl:variable name="text_color"
select="especificacao/infobox/fonte/@cor"/>
<xsl:variable name="text_face"
select="especificacao/infobox/fonte/@face"/>
<xsl:variable name="text_size"
select="especificacao/infobox/fonte/@tamanho"/>

<outputxsl:template name="info1">
<g id="info1" transform="translate(0,0)" visibility="hidden">
<rect width="200" height="16" stroke="gray" stroke-width="1" fill="white"
opacity="0.8"/>
<text id="info10" x="2" y="12" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
</g>
</outputxsl:template>

<outputxsl:template name="info2">
<g id="info2" transform="translate(0,0)" visibility="hidden">
<rect width="200" height="30" stroke="gray" stroke-width="1" fill="white"
opacity="0.8"/>
<text id="info20" x="2" y="12" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
<text id="info21" x="2" y="26" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
</g>
</outputxsl:template>

<outputxsl:template name="info3">
<g id="info3" transform="translate(0,0)" visibility="hidden">
<rect width="200" height="44" stroke="gray" stroke-width="1" fill="white"
opacity="0.8"/>
<text id="info30" x="2" y="12" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
<text id="info31" x="2" y="26" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
<text id="info32" x="2" y="40" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
</g>
</outputxsl:template>

<outputxsl:template name="info4">
<g id="info4" transform="translate(0,0)" visibility="hidden">
<rect width="200" height="58" stroke="gray" stroke-width="1" fill="white"
opacity="0.8"/>
<text id="info40" x="2" y="12" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
<text id="info41" x="2" y="26" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
<text id="info42" x="2" y="40" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
<text id="info43" x="2" y="54" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>
</g>
</outputxsl:template>

<outputxsl:template name="info5">
<g id="info5" transform="translate(0,0)" visibility="hidden">
<rect width="200" height="72" stroke="gray" stroke-width="1" fill="white"
opacity="0.8"/>
<text id="info50" x="2" y="12" font-family="{ $text_face }" font-
size="{ $text_size }" fill="{ $text_color }">-</text>

```

```

<text id="info51" x="2" y="26" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
<text id="info52" x="2" y="40" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
<text id="info53" x="2" y="54" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
<text id="info54" x="2" y="68" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
</g>
</outputxsl:template>

<outputxsl:template name="info6">
<g id="info6" transform="translate(0,0)" visibility="hidden">
<rect width="200" height="86" stroke="gray" stroke-width="1" fill="white" opacity="0.8"/>
<text id="info60" x="2" y="12" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
<text id="info61" x="2" y="26" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
<text id="info62" x="2" y="40" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
<text id="info63" x="2" y="54" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
<text id="info64" x="2" y="68" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
<text id="info65" x="2" y="82" font-family="{ $text_face}" font-size="{ $text_size}" fill="{ $text_color}"> </text>
</g>
</outputxsl:template>

</outputxsl:stylesheet>

</xsl:template>

<xsl:template name="truncate">
<xsl:param name="theString" select="."/>
<xsl:variable name="stringLength" select="string-length($theString)"/>
<xsl:value-of select="substring($theString,0,20)"/>
<xsl:if test="$stringLength > 20">...</xsl:if>
</xsl:template>

</xsl:stylesheet>

```

Apêndice C

```
<xsl:stylesheet version="1.1"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xlink="http://www.w3.org/1999/xlink">
<xsl:output encoding="UTF-8" indent="yes" omit-xml-declaration="yes"
standalone="no"></xsl:output>
<xsl:strip-space elements="*" /><!-- without this line-breaks appear as
text nodes -->
<xsl:template match="/">
function onload(evt){
    var target = evt.getTarget();
    doc = target.ownerDocument();
}

function mouseoverObject(evt,n) {
    var oTarget = evt.target;
    var ginfo = new Array();
    if (oTarget.nodeName() != "text") {
        return false;
    }
    for (i=0; i!=n; i++)
    {
        var Info =
oTarget.parentNode.getElementsByTagName("atrtrunc").item(i);
        ginfo[i] = Info.getFirstChild();
        if (ginfo[i] != null) {ginfo[i] = "-";} else {ginfo[i] =
Info.getFirstChild().nodeValue;}
    }
    showInfo(oTarget,true,n,ginfo,evt.clientX+10,evt.clientY-40);
}

function mouseoutObject(evt,n) {
    var oTarget = evt.target;
    if (oTarget.nodeName() != "text") {
        return false;
    }
    showInfo(oTarget,false,n);
}

function showInfo(oTarget,show,n,ginfo,x,y) {
    var SVGDoc = oTarget.ownerDocument();
    var oToolTip = SVGDoc.getElementById('info'+n);
    var TextL = new Array();
    if (show) {
        for (i=0; i!=n; i++)
        {
            {
                Text[i] = SVGDoc.getElementById("info"+n+i).getFirstChild();
                Text[i].setData(ginfo[i]);
            }
            oToolTip.setAttribute("transform","translate(" + x + "," + y
+ ")");
            oToolTip.setAttribute("visibility","visible");
        }
    }
    else {
        oToolTip.setAttribute("visibility","hidden");
    }
}

<xsl:if test="especificacao/filtros/@presente = 'sim'">
<xsl:for-each select="especificacao/filtros/filtro">
function show<xsl:value-of select="@nome"/> (evt,info)
```

```

{
    var target = evt.getTarget();
    var check = evt.target;
    SVGDoc = target.ownerDocument();
    checked = check.parentNode.getElementsByTagName("rect").item(0);
    ctemp = checked.getFirstChild();
    checked = checked.getFirstChild().nodeValue;

    if (checked == 'yes'){
        <xsl:for-each select="grupoclem">
            ct = SVGDoc.getElementById("<xsl:value-of select='@grupoid'/>");
            ct = ct.getElementsByTagName('num').item(0);
            ct = parseInt(ct.getFirstChild().nodeValue);
            ct++;
            for (i=1;i!=ct;i++)
            {
                elem = SVGDoc.getElementById('<xsl:value-of
select="@grupoid"/>'+i);
                atribContent = elem.getElementsByTagName('atr').item(<xsl:value-of
select="@atributoid"/>);
                atribContent = atribContent.getFirstChild().nodeValue;
                if (atribContent == info)
                {
                    elemFC =
elem.getElementsByTagName('inactivefilters').item(0);
                    elemFCtemp = elemFC.getFirstChild();
                    elemFC = parseInt(elemFC.getFirstChild().nodeValue);
                    c = elemFC+1;
                    elemFCtemp.setData(c);
                    elemAT = elem.getElementsByTagName('use').item(0);
                    elemAT.setAttribute("visibility","hidden");
                }
            }
        </xsl:for-each>
        check.setAttribute("style","fill:white;stroke:black;stroke-
width:1");
        ctemp.setData('no');
    }
    else {
        <xsl:for-each select="grupoclem">
            ct = SVGDoc.getElementById("<xsl:value-of select='@grupoid'/>");
            ct = ct.getElementsByTagName('num').item(0);
            ct = parseInt(ct.getFirstChild().nodeValue);
            ct++;
            for (i=1;i!=ct;i++)
            {
                elem = SVGDoc.getElementById('<xsl:value-of
select="@grupoid"/>'+i);
                atribContent = elem.getElementsByTagName('atr').item(<xsl:value-of
select="@atributoid"/>);
                atribContent = atribContent.getFirstChild().nodeValue;
                if (atribContent == info)
                {
                    elemFC =
elem.getElementsByTagName('inactivefilters').item(0);
                    elemFCtemp = elemFC.getFirstChild();
                    elemFC = parseInt(elemFC.getFirstChild().nodeValue);
                    c = elemFC + 1;
                    elemFCtemp.setData(c);
                    if (c == 0)
                    {elemAT = elem.getElementsByTagName('use').item(0);
                    elemAT.setAttribute("visibility","visible");}
                }
            }
        </xsl:for-each>
    }
}

```

```

        check.setAttribute("style", "fill:blue;stroke:black;stroke-
width:1");
        ctemp.setData('yes');
    }
}
</xsl:for-each>
</xsl:if>

<xsl:for-each select="especificacao/elementos/elemento">
function showrelated<xsl:value-of select="@grupoid"/> (evt,info)
{
    var target = evt.getTarget();
    var check = evt.target;
    SVGDoc = target.getOwnerDocument();
    checked = check.parentNode.getElementsByTagName("use").item(0);
    ctemp = checked.getFirstChild();
    checked = checked.getFirstChild().nodeValue;

    if (checked == 'yes'){
    <xsl:for-each select="comandos/mousedown/grupoelem">
    ct = SVGDoc.getElementById("<xsl:value-of select="@grupoid"/>");
    ct = ct.getElementsByTagName('num').item(0);
    ct = parseInt(ct.getFirstChild().nodeValue);
    ct++;
    for (i=1;i!=ct;i++)
    {
        elem = SVGDoc.getElementById('<xsl:value-of
select="@grupoid"/>'+i);
        atribContent = elem.getElementsByTagName('atr').item(<xsl:value-of
select="@atributoid"/>);
        atribContent = atribContent.getFirstChild().nodeValue;
        if (atribContent == info)
        {
            elemAT = elem.getElementsByTagName('use').item(0);
            elemAT.setAttribute("stroke", "<xsl:value-of
select="@stroke_cor"/>");
            elemAT.setAttribute("fill", "<xsl:value-of select="@cor"/>");
            elemAT = elemAT.getFirstChild();
            elemAT.setData('no');
        }
    }
    </xsl:for-each>
    check.setAttribute("stroke", "<xsl:value-of
select="comandos/mousedown/parametros/@stroke_cor"/>");
    check.setAttribute("fill", "<xsl:value-of
select="comandos/mousedown/parametros/@cor"/>");
    ctemp.setData('no');
    }
    else {
    <xsl:for-each select="comandos/mousedown/grupoelem">
    ct = SVGDoc.getElementById("<xsl:value-of select="@grupoid"/>");
    ct = ct.getElementsByTagName('num').item(0);
    ct = parseInt(ct.getFirstChild().nodeValue);
    ct++;
    for (i=1;i!=ct;i++)
    {
        elem = SVGDoc.getElementById('<xsl:value-of
select="@grupoid"/>'+i);
        atribContent = elem.getElementsByTagName('atr').item(<xsl:value-
of select="@atributoid"/>);
        atribContent = atribContent.getFirstChild().nodeValue;
        if (atribContent == info)
        {
            elemAT = elem.getElementsByTagName('use').item(0);

```

```

        elemAT.setAttribute("stroke", "<xsl:value-of
select='@stroke_cor_original' />");
        elemAT.setAttribute("fill", "<xsl:value-of
select='@cor_original' />");
        elemAT = elemAT.getFirstChild();
        elemAT.setData('yes');
    }
}
</xsl:for-each>
    check.setAttribute("stroke", "<xsl:value-of
select='representacao/@stroke_cor' />");
    check.setAttribute("fill", "<xsl:value-of
select='representacao/@cor' />");
    ctemp.setData('yes');
}
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```